Xin Yang

xy213

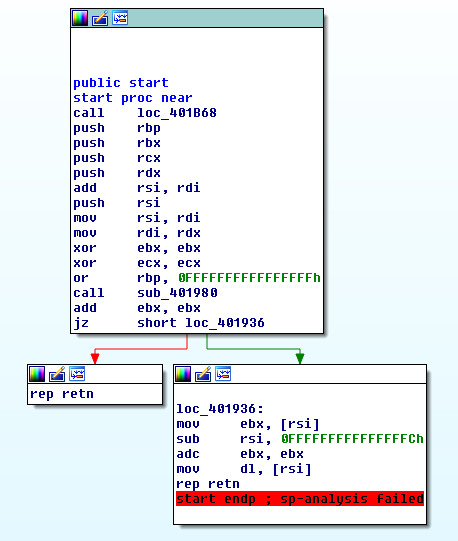
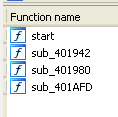
Midterm Malware Report

**Unpack:**

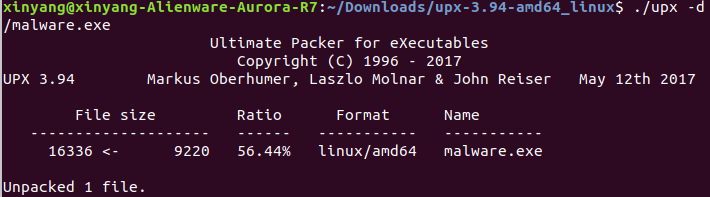
The file command shows this malware is a 64-bit ELF executable, same as VirusTotal.



If I open this file as ELF64, there are only four functions and from the graph view we can see it’s completely not what we want.

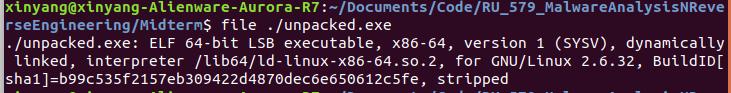


In this case, I ndeed to unpack this malware first. Since it’s packed by UPX. The latest release of UPX provides a convenient way to decompress by using the command “upx -d malware.exe”, and it successfully unpacked this malware.



After unpacked, I can open it IDA Pro and the results are similar to the Homework 4.

But as the file command tells, it’s still a stripped file, which means when the program is compiled, the writer deleted many libraries and symbol tables to save space occupation and hide essential information. So here lots of function names are replaced by addresses.



But with the reference of Homework4, we 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 compare 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 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.

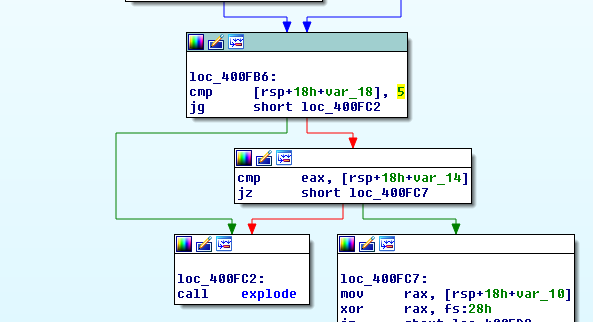
Phase 2 Answer:

**Any sequence satisfy 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 in 8 cases. In each case, the program will calculate a pre-set number using add and sub, and keep the value in eax, then compare it with [rsp+18h+var\_14], which is the second input number, explode if they are not equal. There are eight cases and the first input number is used for case switch. So we can get all eight corresponding pairs, but this program will compare the case number, which is [rsp + 18h + var\_18] with 5, and explode if bigger than 5.



So here we only have six pairs of answers.

Phase 3 Answer:

case 0: **0 -556**

case 1: **1 -1501**

case 2: **2 -554**

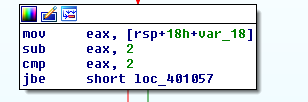
case 3: **3 -866**

case 4: **4 0**

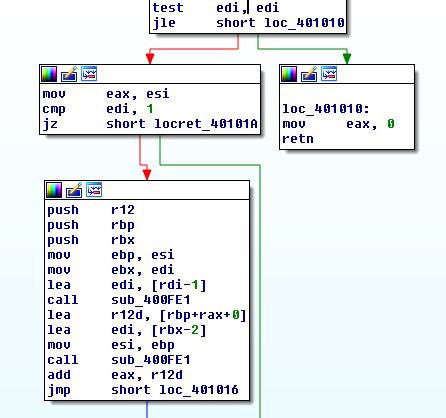
case 5: **5 -866**

**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 sub\_400FE1 function, in this function this program will recursively call itself 53(the number of nodes having values greater than 0 by transfer the recursive into a binary tree, starting at root 8) times and each time add the value of the second input number, which is stored in esi, ebp, and rbp,

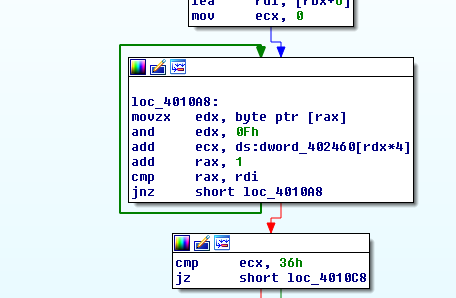
r12d = [rbp + rax + 0] = 3 + [rax], where [rax] is the returned value of the sub function. So this program will add the second number for 53 times to itself, then leave the recursion and compare it with the first input number, i.e., compare the first input with 54 times second input. This phase will be defused if they 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, 6’s ASCII, and the inner loop will compare the pointer with the last number in the input, and break the loop if equals. We can assume that this program will add 1 to the previous one until comes to the last digit, which is 54. So the answer should start from 49 to 54 in ASCII, in number, the result should be 123456.

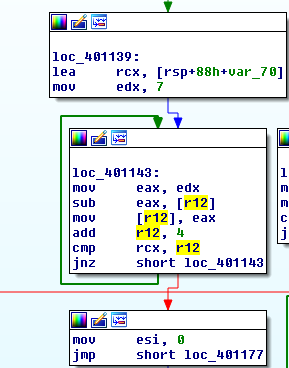


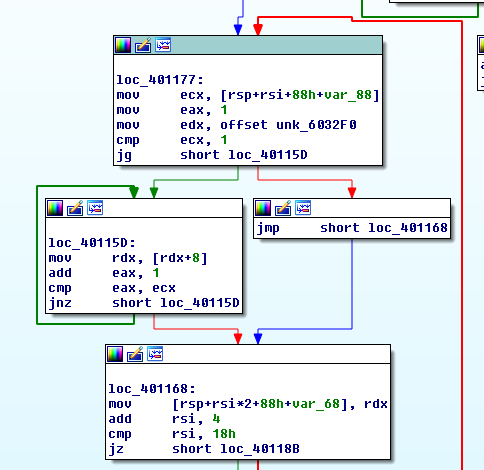
Phase 5 Answer:

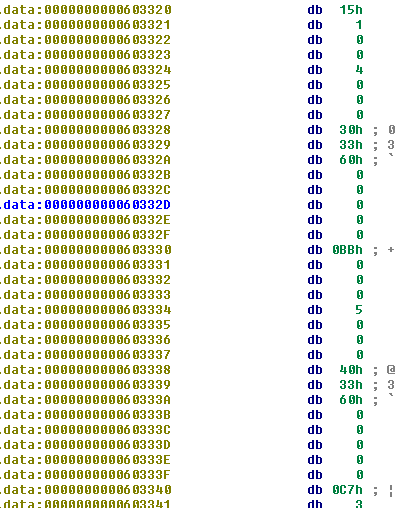
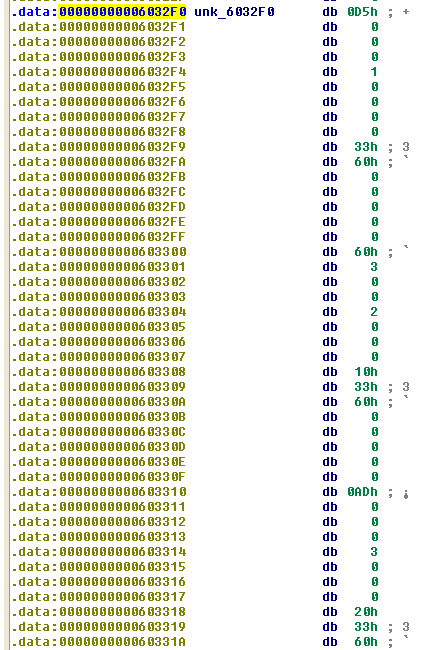
**123456**

**Phase 6:**

The last phase also calls the readsixnumber() function. And from the loop after, the input should be six numbers and all smaller or equal to six. Then the program will store 7 minus input to replace the original value in r12.

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



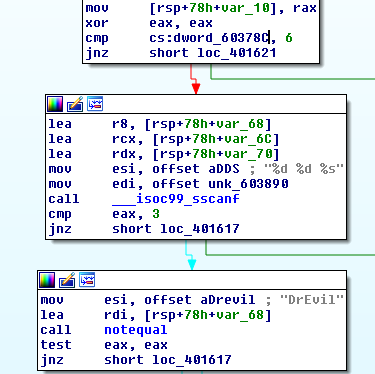
Similarly as homework, this program will later verify if the numbers are in ascending orders. But since the sequence is arranged by the recalculated value, the input numbers should be in descending order. By comparing the numbers of the values each node represents, we got the order: 1 5 3 6 2 4.

Phase 6 Answer:

**1 5 3 6 2 4**

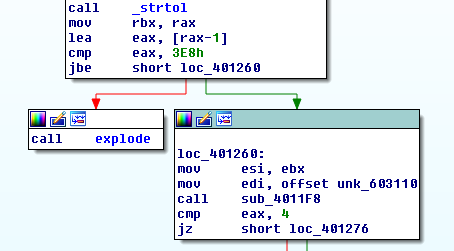
**Secret Phase:**

By looking at the phase\_defused phase, we can see this stage will be activated after put a string behind two numbers. Thus phase three and four are possible, but phase three will explode if the arguments are more than two, so the secret phase can only happen together with phase 4.

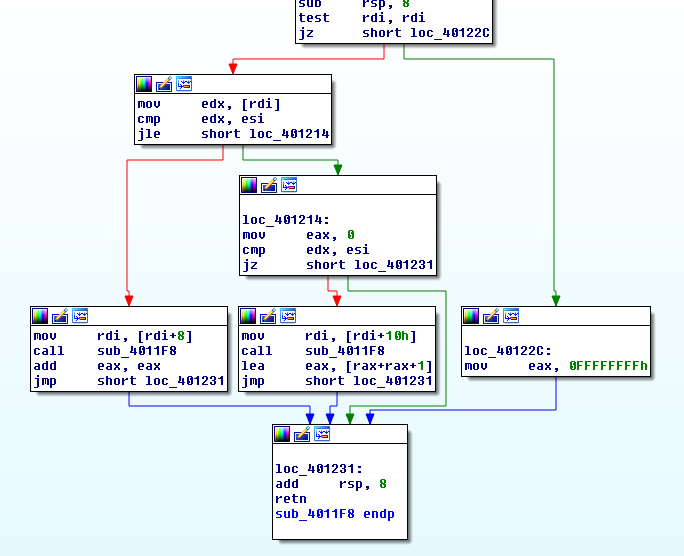
Since the string is equals the length of 6, from the hint we can find the code “DrEvil” after phase 4 is the entry to the secret phase. The address of secret phase is sub\_401236, and the result should return 4.

Here the input number should also be smaller than 3E8h(1000).

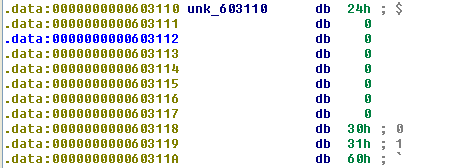
603150.



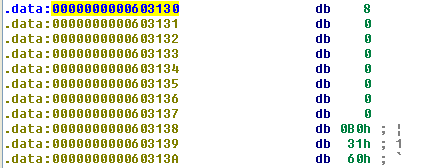
From the inner recursion function we can see that the recursion will return a value under two circumstances: First: rdi is 0, this will return -1. Second: edx = esi, which means user input equals the current argument. And this function will call a recursion also under two circumstances: First: the argument is bigger than input: call the recursion with new argument to be the one having address + 8, return 2\*result. Second: the argument is smaller than user input: call recursion with new argument to be the one having address + 16, and return 2\*result + 1.



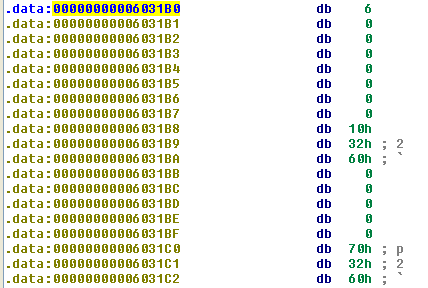
Since the final returned value is 4, which is even, so the last returned formula must be 2\*result, which means the user input is smaller than the argument, which is 24h at 603110h.



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



Next the target returned value should be 2/2 = 1. This time the value should be smaller than user input to get a 2\*0+1 formula. Tracing back the 6031B0h (stored in [603130+8h]), we can get the value 6. Here 6 means the input should be greater than 6, so we now have the range of input is between 6 to 8.



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



Secret Phase Answer:

**7**