

Windows - Task 4 Report

Name : Ishayu Potey

College : VJTI SY Btech - Electronics

Unique ID : LTXBGRJ6

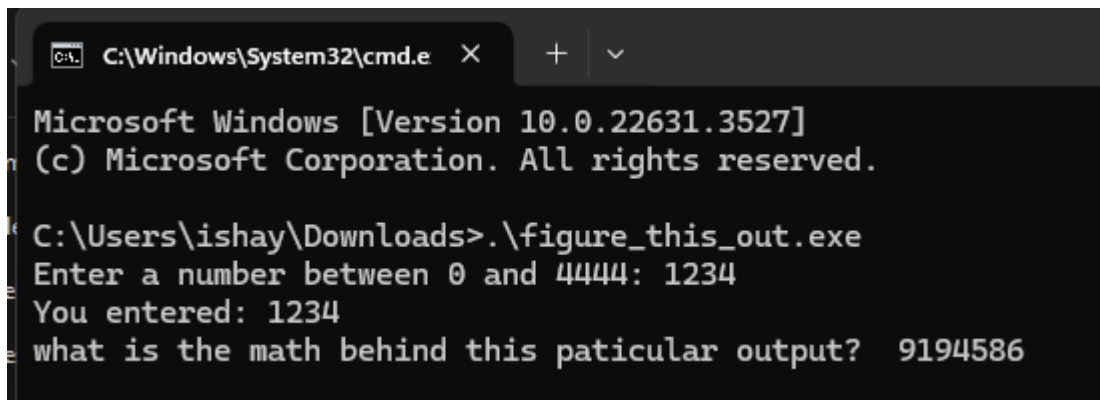
Aim : Reverse engineer the `figure_this_out.exe` file and uncover the mathematical calculations and operations behind the output generation.

We have to identify the Mathematical Algorithm behind the outputs and create a similar program in Python

In this we have an **Windows Portable Executable** File (PE) with 64-bit Architecture

```
(kali㉿kali)-[~/Desktop]
$ file figure_this_out.exe
figure_this_out.exe: PE32+ executable (console) x86-64, for MS Windows, 6 sections
```

Running the `figure_this_out.exe` in Windows CMD



```
C:\Windows\System32\cmd.e  X  +  v
Microsoft Windows [Version 10.0.22631.3527]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ishay\Downloads>.\figure_this_out.exe
Enter a number between 0 and 4444: 1234
You entered: 1234
what is the math behind this paticular output?  9194586
```

- We observe that it takes a restricted input from 0 to 4444
- based on that it produces some output
- Lets Try Basic Brute Force Method we take multiple inputs for their respective outputs
- To get the Line Equation for this algorithm in Python.

```
Equation_Task4.py
C: > Users > ishay > OneDrive > Desktop > LTXBGRJ6 Windows Task-4 > Equation_Task4.py > ...
1  import numpy as np
2
3  i = [0, 1234, 1111, 4444, 3500, 2222, 500, 1, 3000, 4000, 1500]
4  o = [52, 9194586, 7458195, 118703736, 73664552, 29728190, 1523552, 105, 54141052, 96188052, 13570552]
5
6  coeff = np.polyfit(i, o, 2)
7  eqn = np.poly1d(coeff)
8
9  test_inp = np.arange(0, 4445, 10)
10
11 test_out = eqn(test_inp)
12
13 print("Final Equation:\n", eqn)
14
15 print("Output values for test inputs:")
16 for X, Y in zip(test_inp, test_out):
17     print(f"For X = {X}, Y is {Y:.2f}")
18
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS
[Running] python -u "c:\Users\ishay\OneDrive\Desktop\LTXBGRJ6 Windows Task-4\Equation_Task4.py"
Equation of the fitted curve:
2
6 x + 47 x + 52
Output values for test inputs:
For X = 0, Y is 52.00
For X = 10, Y is 1122.00
For X = 20, Y is 3392.00
For X = 30, Y is 6862.00
For X = 40, Y is 11532.00
For X = 50, Y is 17402.00
```

While coding this I realized it was a 2 - Degree Polynomial (Quadratic) .

- We took some bunch of raw inputs and their outputs from CMD
- Tried to Fit a Polynomial Equation and curve against it.
- Since the answer was not coming accurate i tried to take bunch of test inputs with For Loop and put it against our polynomial Equation.
- After Running the code we obtain a Quadratic Equation : **$6x^2 + 47x + 52$**

Writing the Solution.py

```
import numpy as np

x = int(input("Enter the value of x: "))

if (0 ≤ x ≤ 4444 ):

    y = 6*x**2 + 47*x + 52
    print(y)

else :
    print("YOU MUST OBEY THE RANGE")
```

I have verified this and its working well

I tried doing it IDA Decompiler though the approach was harder than brute force

- We then Open the `figure_this_out.exe` File in IDA Decompiler.
- Opening the Pseudocode for following executable binary we get this code

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
    unsigned int v3; // eax
    int v5; // [rsp+2Ch] [rbp-3Ch]
    int i; // [rsp+3Ch] [rbp-2Ch]
    int v7; // [rsp+40h] [rbp-28h]
    unsigned int v8; // [rsp+44h] [rbp-24h]
    unsigned int v9; // [rsp+48h] [rbp-20h]
    unsigned int v10; // [rsp+4Ch] [rbp-1Ch]
    unsigned int v11; // [rsp+58h] [rbp-10h]
    int v12[2]; // [rsp+60h] [rbp-8h] BYREF

    v12[1] = 0;
    v12[0] = 0;
    sub_1400019E0("Enter a number between 0 and 4444: ", argv, envp);
    sub_140001A50("%d", v12);
    sub_1400019E0("You entered: %d\n", (unsigned int)v12[0]);
    v11 = 78123712;
    v10 = 0;
    v9 = v12[0] + 2;
    v8 = 6 * v12[0];
    v7 = v12[0] + 8;
    sub_1400019E0("what is the math behind this paticular output?");
    for ( i = 1; ; ++i )
    {
        v3 = add(2LL, 3LL);
        if ( i > (int)add(5LL, v3) )
            break;
        v5 = add(v9, v8);
        sub_140001020(v8 + 7, (unsigned int)(v7 + 12));
        v11 = sub_1400010B0(v11, 0LL);
        v10 = sub_140001780(v9 + 3, i + v8) + v5;
    }
    sub_1400013E0(v11);
    sub_1400019E0(" %d\n", v10);
    return 0;
}
```

Analyzing Assembly and Pseudocode

- After Hours of watching the code I realized
- The Pseudocode had many functions and most of them were redundant.
- The Only function which was doing the work was `sub_140001780`
In this function there were two For loops and one of them was redundant since it returned v8 value so, when we decode it properly and calculating it by writing variables on pen and paper.
- I got this formula

$$v10 = (6 * v12 * v12) + (47 * v12) + 52$$

It is similar to one which we got earlier : $6x^2 + 47x + 52$

I Wrote python script for same

```
def sub_1400019E0(prompt, *args):  
    print(prompt % args)  
  
def sub_140001A50(fmt, value):  
    value = int(input(fmt))  
    return value  
  
def sub_1400013E0(x):  
    print(x)  
  
v12 = 0  
sub_1400019E0("Enter a number between 0 and 4444: ")  
v12 = sub_140001A50("", v12)  
sub_1400019E0("You entered: %d\n" % v12)  
  
v10 = (6 * v12 * v12) + (47 * v12) + 52  
  
sub_1400019E0(" %d\n" % v10)
```

EQUATION : $6x^2 + 47x + 52$

Thank You