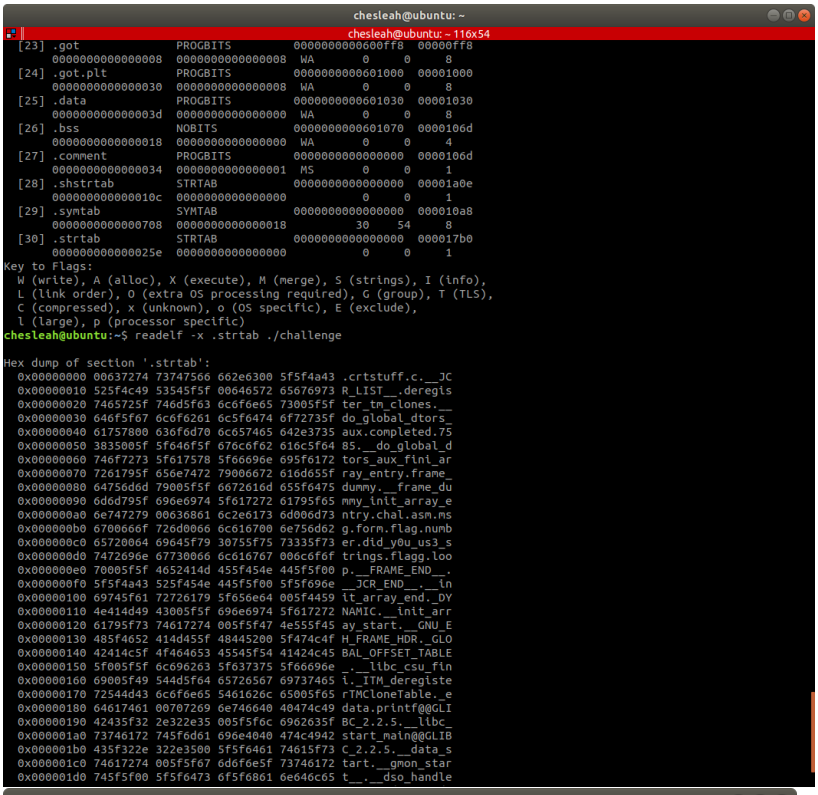
Practical Cyber Security Fundamentals
Assignment 7: Reversing I

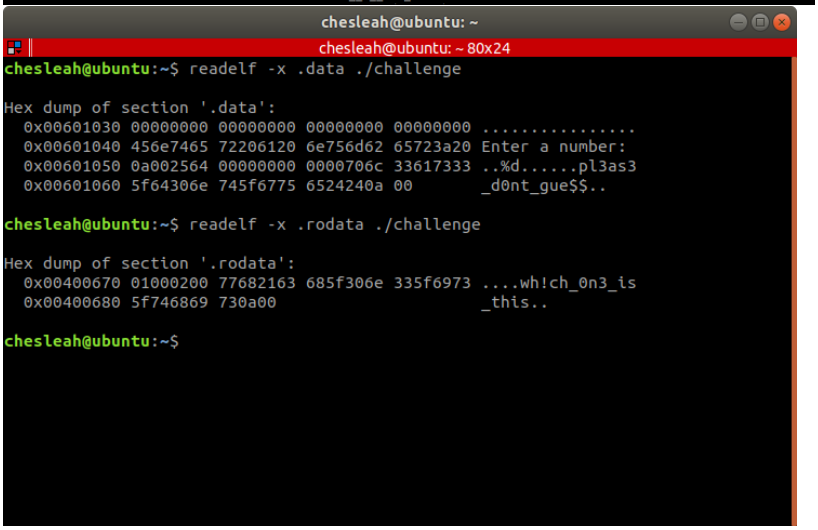
Overview
Assignment: Reversing I
Student: Chesleah Kribs
Date: March 12, 2019

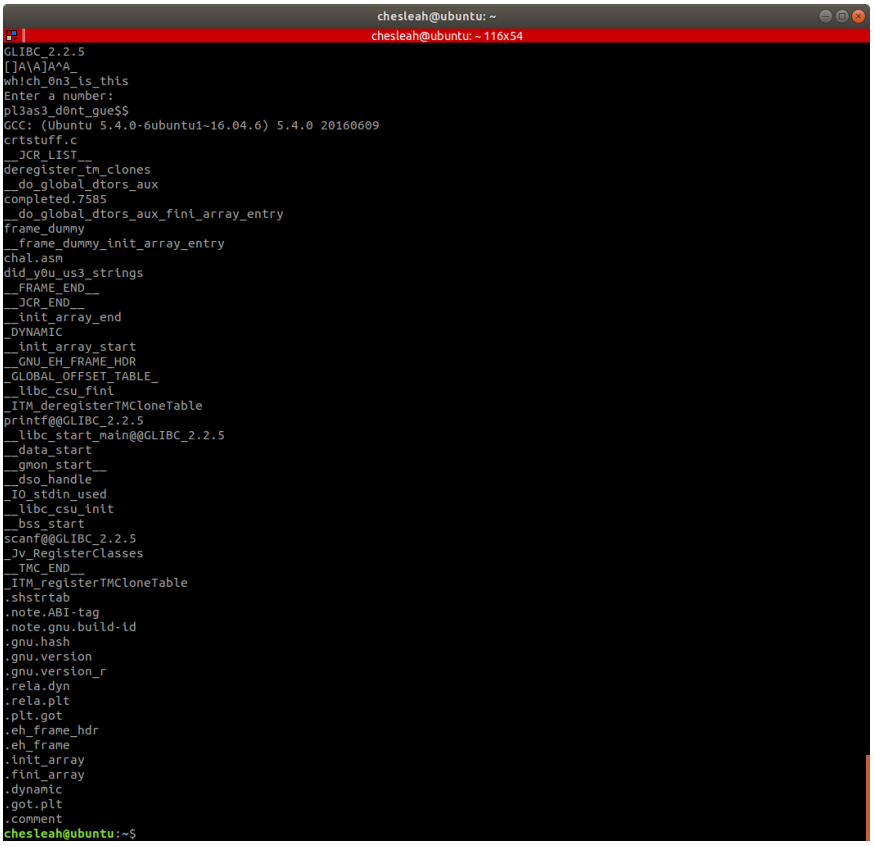
Problem 1
Problem: Command Journal

Command	**Description**
Mov eax, [ebx]	Move the bytes in memory at the address contained in ebx to eax
Mov [var], ebx	Move the contents of ebx into the bytes at memory address of var
Mov BYTE PTR [ebx], 2	Move 2 into the single byte at the address stored in EBX
Push eax	Push eax onto the stack
Pop edi	Pop the top element of the stack into EDI
Lea edi, [ebx + 4 * esi]	The quantity EBX + 4 * ESI is placed in EDI
Add eax, 10	Eax = eax + 10
Sub al, ah	Al = al – ah
dec eax	Subtract one from the contents of eax
Xor eax, eax	Clears out eax, sets to 0
Jmp label	Jumps to the label

Problem 2
Problem: Name that Section
Analysis: This problem is a evaluation of where in the memory certain strings are located. You could use the strings command to find these.
Plan: I knew I wanted to approach this problem with using strings -n 8 and readelf -S to find the specific sections.
Solution: In order to solve this problem I originally did strings -n 8 ./challenge. This located three unusual strings: "whi!ch_0n3_is_this, pl3as3_d0nt_gue$$, and did_y0u_us3_strings." After doing the -S flag on the readelf command, we could see there are many sections. I did a guess.. yeah sorry, and found the please don't guess in .data, and .rodata had which one is this, and .strtab had the flag: did you use strings.
flag: flag{did_y0u_us3_strings}


chesleah@ubuntu: ~
chesleah@ubuntu: ~ 116x54
[23] .got PROGBITS 0000000000600ff8 00000ff8
0000000000000008 0000000000000008 WA 0 0 8
[24] .got.plt PROGBITS 0000000000601000 00001000
0000000000000030 0000000000000008 WA 0 0 8
[25] .data PROGBITS 0000000000601030 00001030
000000000000003d 0000000000000000 WA 0 0 8
[26] .bss NOBITS 0000000000601070 0000106d
0000000000000018 0000000000000000 WA 0 0 4
[27] .comment PROGBITS 0000000000000000 0000106d
0000000000000034 0000000000000001 MS 0 0 1
[28] .shstrtab STRTAB 0000000000000000 00001a0e
000000000000010c 0000000000000000 0 0 1
[29] .symtab SYMTAB 0000000000000000 000010a8
0000000000000708 0000000000000018 30 54 8
[30] .strtab STRTAB 0000000000000000 000017b0
000000000000025e 0000000000000000 0 0 1
Key to Flags:
W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
L (link order), O (extra OS processing required), G (group), T (TLS),
C (compressed), x (unknown), o (OS specific), E (exclude),
l (large), p (processor specific)
chesleah@ubuntu:~$ readelf -x .strtab ./challenge
Hex dump of section '.strtab':
0x00000000 00637274 73747566 662e6300 5f5f4a43 .crtstuff.c.__JC
0x00000010 525f4c49 53545f5f 00646572 65676973 R_LIST__.deregis
0x00000020 7465725f 746d5f63 6c6f6e65 73005f5f ter_tm_clones.__
0x00000030 646f5f67 6c6f6261 6c5f6474 6f72735f do_global_dtors_
0x00000040 61757800 636f6d70 6c657465 642e3735 aux.completed.75
0x00000050 3835005f 5f646f5f 676c6f62 616c5f64 85.__do_global_d
0x00000060 746f7273 5f617578 5f66696e 695f6172 tors_aux_fini_ar
0x00000070 7261795f 656e7472 79006672 616d655f ray_entry.frame_
0x00000080 64756d6d 79005f5f 6672616d 655f6475 dummy.__frame_du
0x00000090 6d6d795f 696e6974 5f617272 61795f65 mmy_init_array_e
0x000000a0 6e747279 00636861 6c2e6173 6d006d73 ntry.chal.asm.ms
0x000000b0 6700666f 726d0066 6c616700 6e756d62 g.form.flag.numb
0x000000c0 65720064 69645f79 30755f75 73335f73 er.did_y0u_us3_s
0x000000d0 7472696e 67730066 6c616767 006c6f6f trings.flagg.loo
0x000000e0 70005f5f 4652414d 455f454e 445f5f00 p.__FRAME_END__.
0x000000f0 5f5f4a43 525f454e 445f5f00 5f5f696e __JCR_END__.__in
0x00000100 69745f61 72726179 5f656e64 005f4459 it_array_end._DY
0x00000110 4e414d49 43005f5f 696e6974 5f617272 NAMIC.__init_arr
0x00000120 61795f73 74617274 005f5f47 4e555f45 ay_start.__GNU_E
0x00000130 485f4652 414d455f 48445200 5f474c4f H_FRAME_HDR._GLO
0x00000140 42414c5f 4f464653 45545f54 41424c45 BAL_OFFSET_TABLE
0x00000150 5f005f5f 6c696263 5f637375 5f66696e _.__libc_csu_fin
0x00000160 69005f49 544d5f64 65726567 69737465 i._ITM_deregiste
0x00000170 72544d43 6c6f6e65 5461626c 65005f65 rTMCloneTable._e
0x00000180 64617461 00707269 6e746640 40474c49 data.printf@@GLI
0x00000190 42435f32 2e322e35 005f5f6c 6962635f BC_2.2.5.__libc_
0x000001a0 73746172 745f6d61 696e4040 474c4942 start_main@@GLIB
0x000001b0 435f322e 322e3500 5f5f6461 74615f73 C_2.2.5.__data_s
0x000001c0 74617274 005f5f67 6d6f6e5f 73746172 tart.__gmon_star
0x000001d0 745f5f00 5f5f6473 6f5f6861 6e646c65 t__.__dso_handle


chesleah@ubuntu: ~
chesleah@ubuntu: ~ 80x24
chesleah@ubuntu:~$ readelf -x .data ./challenge
Hex dump of section '.data':
0x00601030 00000000 00000000 00000000 00000000 ....
0x00601040 456e7465 72206120 6e756d62 65723a20 Enter a number:
0x00601050 0a002564 00000000 0000706c 33617333 ..%d......pl3as3
0x00601060 5f64306e 745f6775 6524240a 00 _d0nt_gue$$..
chesleah@ubuntu:~$ readelf -x .rodata ./challenge
Hex dump of section '.rodata':
0x00400670 01000200 77682163 685f306e 335f6973 ....wh!ch_0n3_is
0x00400680 5f746869 730a00 _this..
chesleah@ubuntu:~$



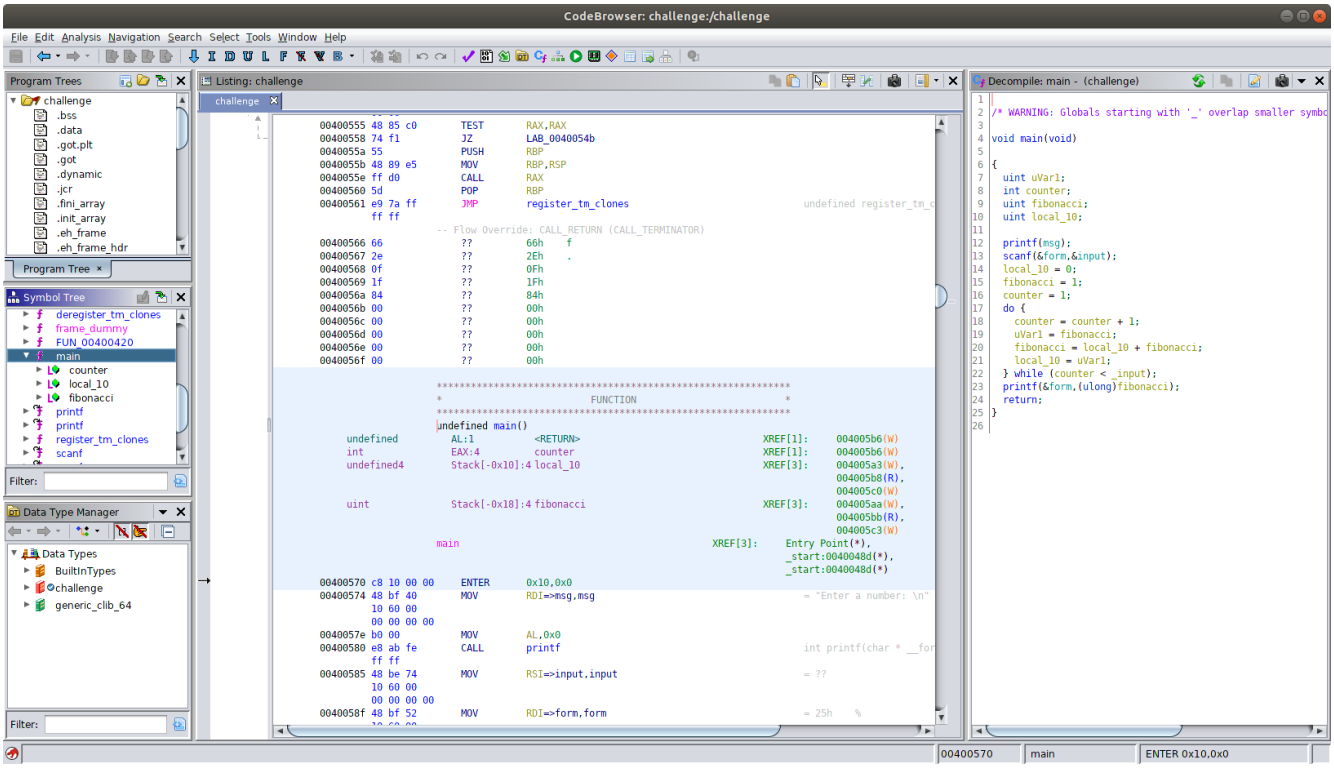
Problem 3

Problem: Name that algorithm

Analysis: Upon analysis, I could see that this program requires input and in order to reverse it and solve for the flag, I had to see what the number output ends up being. To do this, I use Ghidra to evaluate the main function.

Plan: I plan to always use Ghidra because it gives me C code to work with and I prefer this over any assembly anyday.

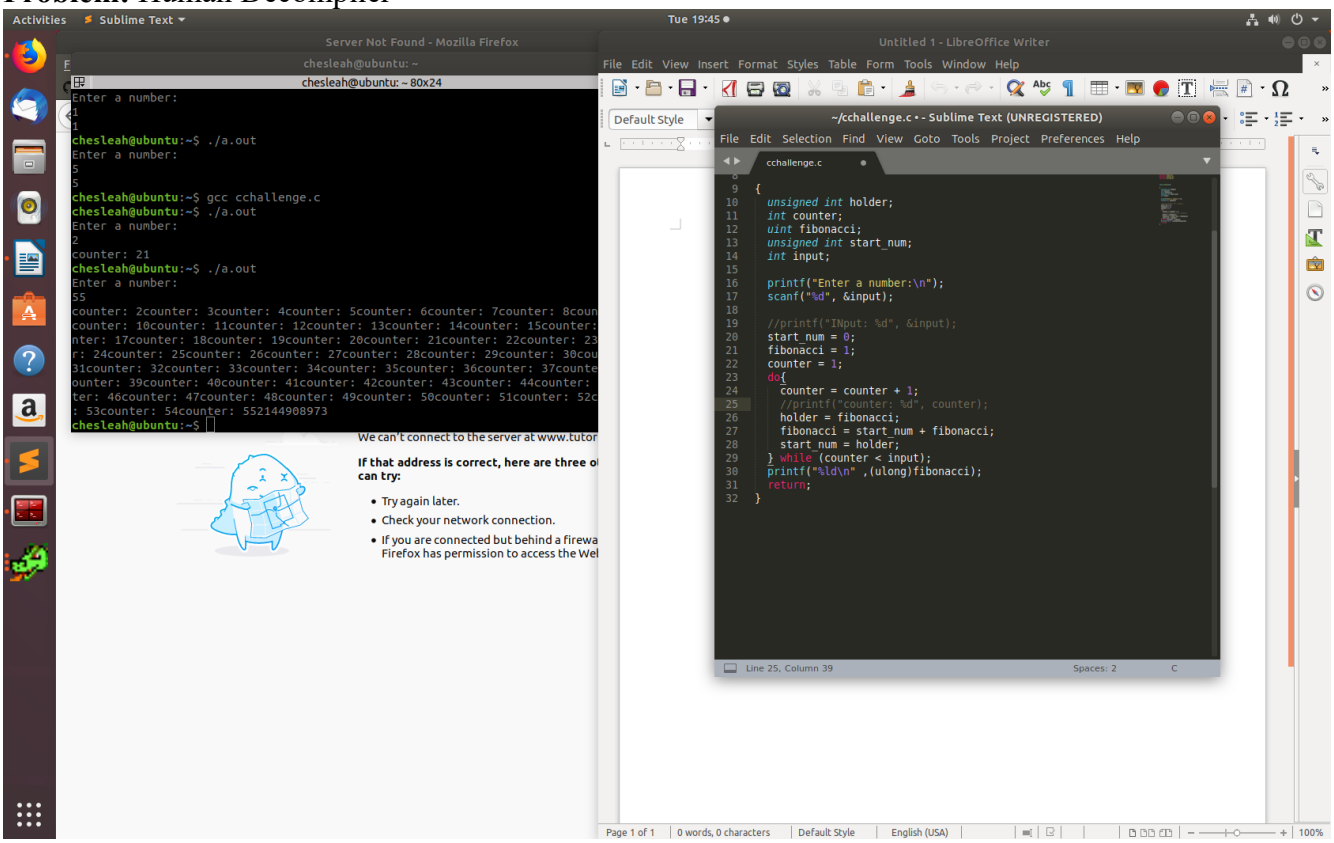
Solution: TO solve this problem, I pushed this into Ghidra. When doing this, I isolated the main function. When I do this, I see the C code. I see that there is some form of a counter, which I name counter because I see it keeps getting incremented. Then I switch the scanf second argument to input, which changes the do while loop to see if the counter is less than the inputed number. I see it starts with a number that is at 1 and then is added to another counter to increment this. After this, I could tell this was a Fibonacci sequence.



flag: flag{fibonacci}

Problem 4

Problem: Human Decompiler



Problem 5

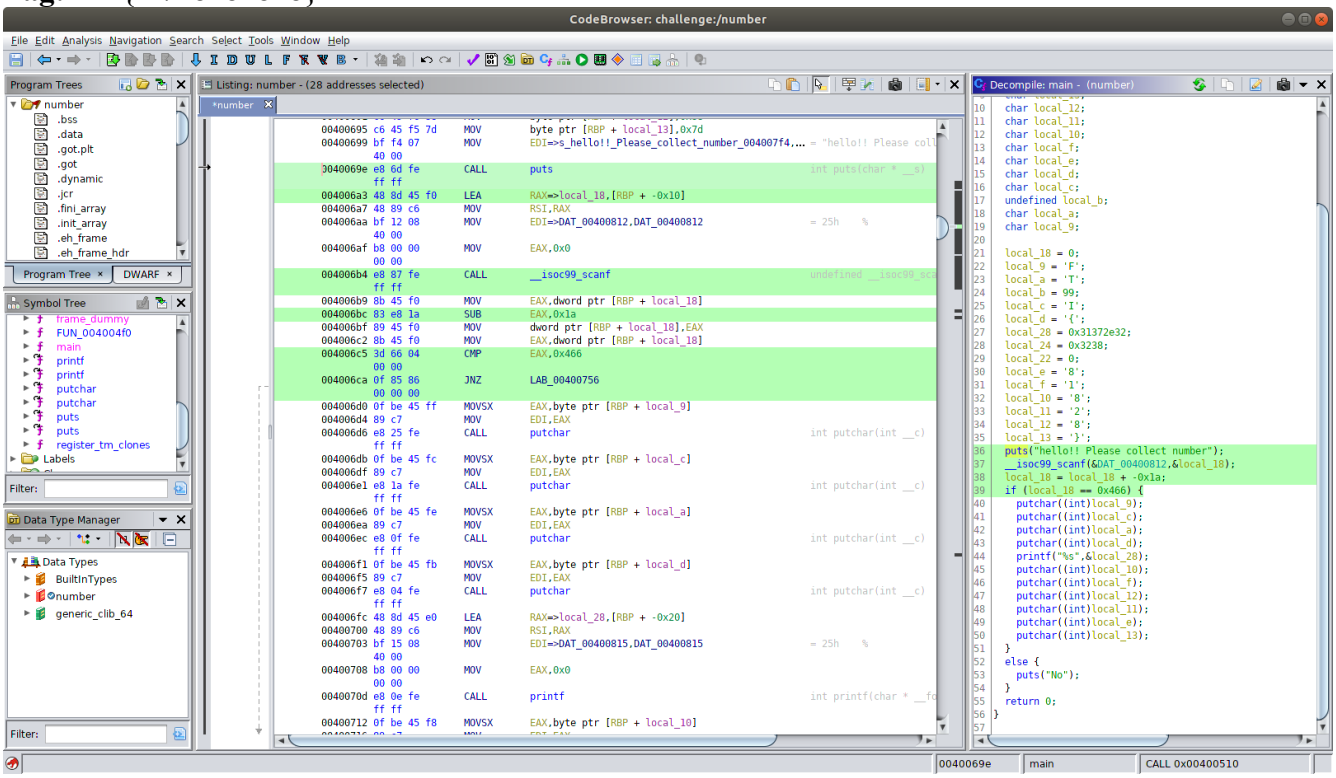
Problem: Number

Analysis: To Analyse this, we will use reverse tools like Radare2 and Ghidra in order to figure out how the binary is working.

Plan: When I ran the program, I noticed that many numbers were not the correct number, so I have to use Ghidra and Radare2 to isolate the number that is proper.

Solution: For the solution, We see in main that there is sub eax, 0x1a which is 26 in hex. We then see that there is a cmp eax, 0x466 which is 1126 in hex. To do this we know that we have to do 1126 + 26 which is 1152. Which is the correct answer when input.

flag: FIT{2.718281828}



Problem 6

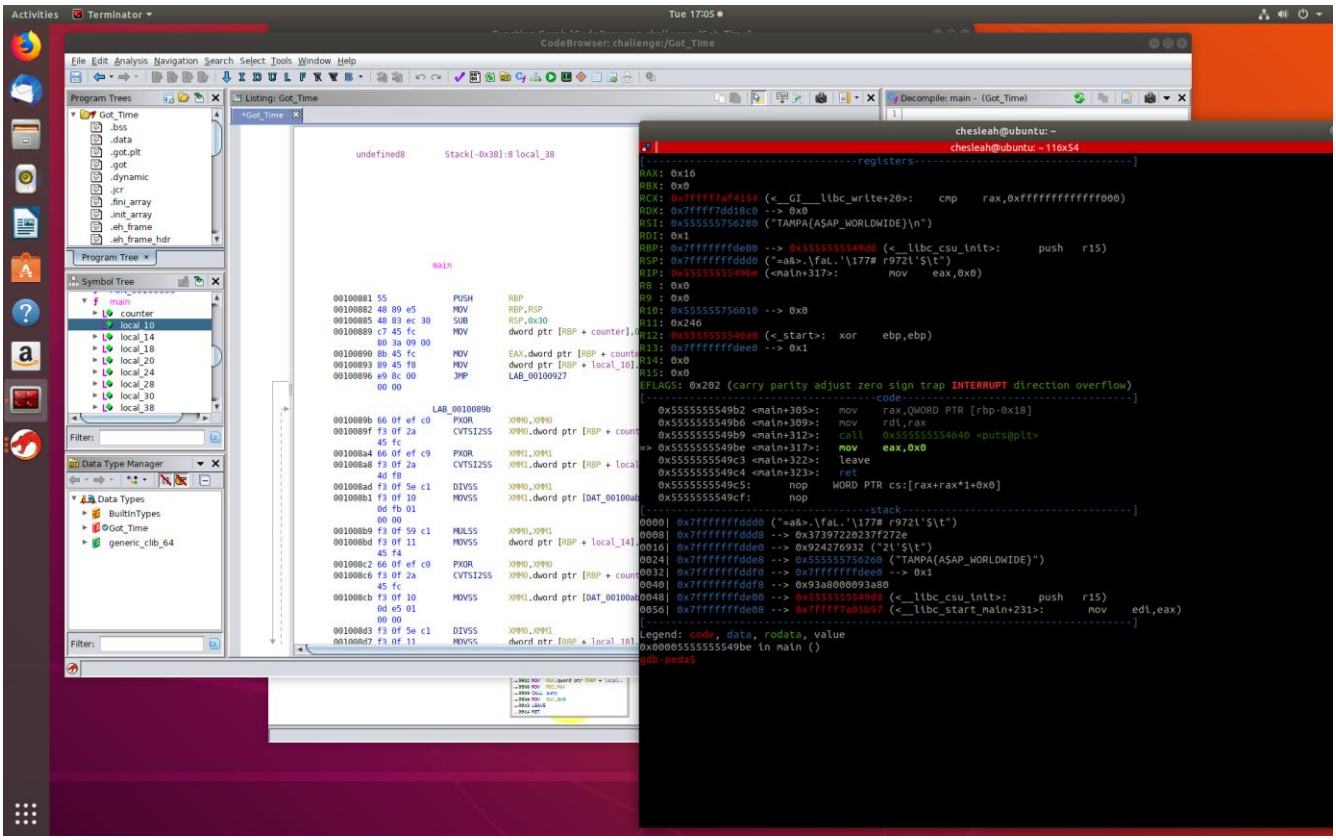
Problem: Got Time?

Analysis: Upon analyzing the program, it seems there is a major timing issue. And I don't have time for dat.

Plan: My plan is to analyze the binary with Ghidra. See the control flow of the binary and try to follow it and solve it.

Solution: In order to solve the problem, I had to use our friend GDB the handy debugger. After I was able to analyze the binary, I noticed that in main, there was a very involved while loop, which was the cause for the "time." I saw that eventually it went to a xorencrypt function. If I am able to jump over the while loop, I could catch the xorencrypt function. By running GDB, I was able to skip over the specific instruction (the jump to the while loop) and then place all the bytes correctly to then call the xor encrypt function to figure out the flag!

flag: TAMPA{A$AP_WORLDWIDE}



Problem 7

Problem: Arrrrrrrrgs

Analysis: To analyze this, I noticed there were far too many arguments to try to follow the control flow, however by recognizing the structure of the arguments, I can use math skills to solve the hex numbers to get a flag for <HACKED>

Plan: My plan is to analyze the binary with Ghidra. See the control flow of the binary and try to follow it and solve it. (It kinda worked)

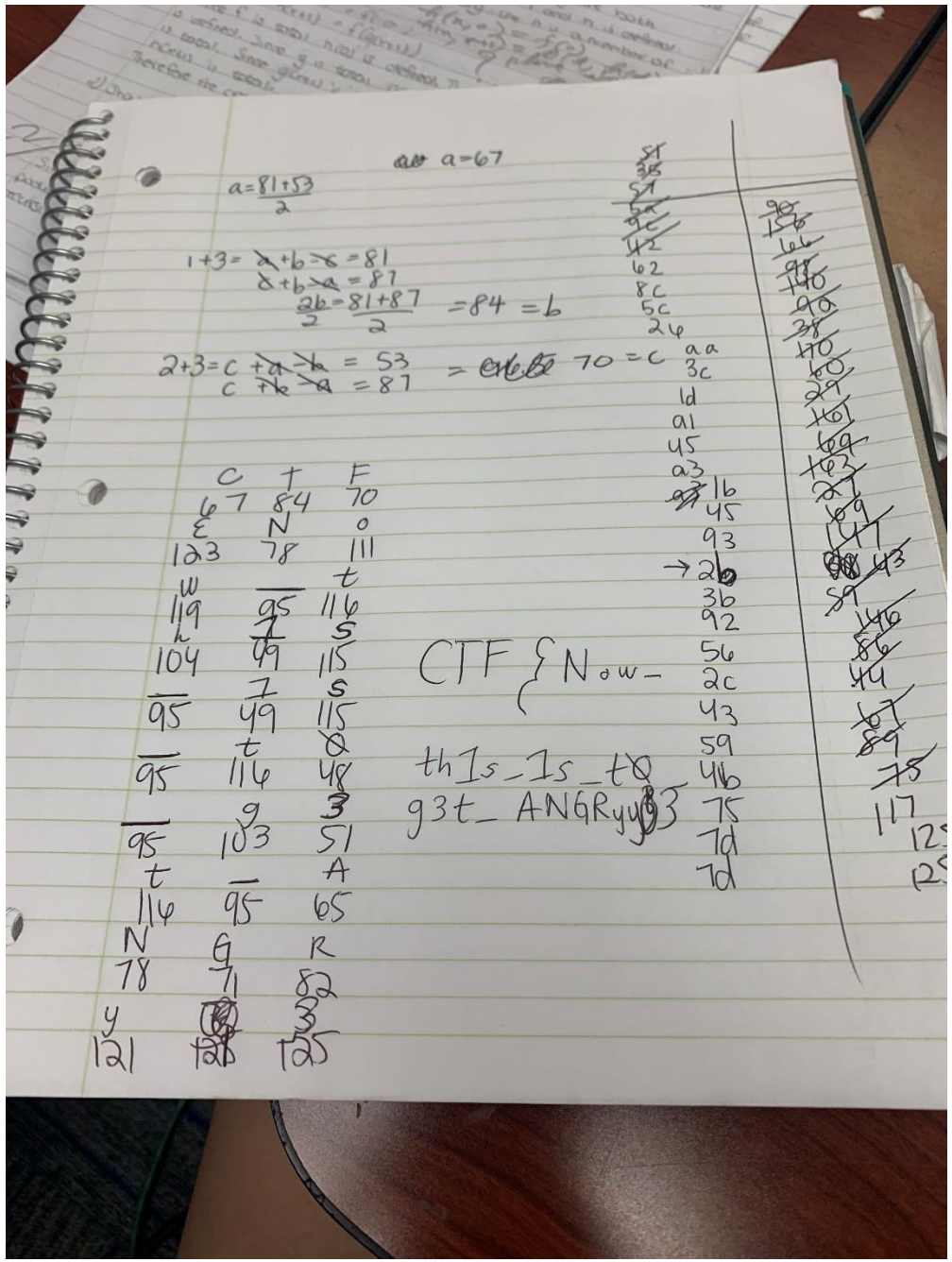
Solution: To solve this problem, I recognized that this was a multifaceted if statement problem. I saw that there was an equation that was equivalent to a specific hex value. Once I was able to see the formula and how it is positioned, I could solve this problem. First, it is a + b -c then a-b+c and finally, -a + b + c. This could be used to find the values for each once setting these to their decimal representations. I went to a linear equation calculator and put in the coefficients and was able to solve for each formula.

I received these values:

67 84 70 123 78 111 119 95 116 104 49 115 95 49 115 95 116 48 95 103 51 116 95 65 78 71 82 121 21 125.

When I translated this to an ascii table I was able to get the flag.

flag: CTF{Now_th1s_1s_t0_g3t_ANGRyy}


a=67
a=81+53/2
1+3= a+b-c=81
a+b-a=87
2b=81+87/2 =84 =b
2+3=c+a-b=53
c+b-a=87 =70=c
CTF{N.ow_
th1s_1s_t0
g3t_ANGRyy}