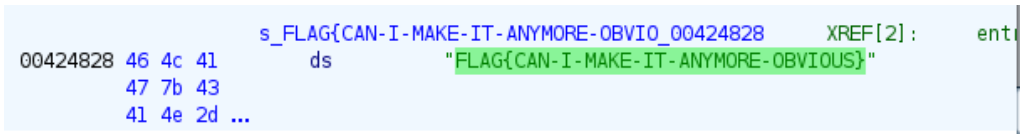
Reverse Engineering

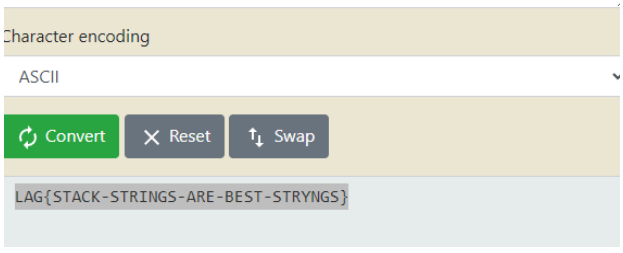
Task 1 Basic Malware RE;

I used Ghidra for all of the tasks here. For *task 2* I double clicked PTR_s_FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIO_00432294 which took me to a place where I can see the plaintext flag FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIOUS}



For *Task 3* I scrolled down until I saw the md5_hash function where I see a collection of local_ ascii char values. I took each of them and put them in [Rapid Tables'](#) website where I was able to build the flag up.

```
4c 41 47 7b 53 54 41 43 4b 2d 53 54 52 49 4e 47 53 2d 41 52 45
2d 42 45 53 54 2d 53 54 52 59 4e 47 53 7d
```



The ‘F’ is missing because the first char is in plaintext.

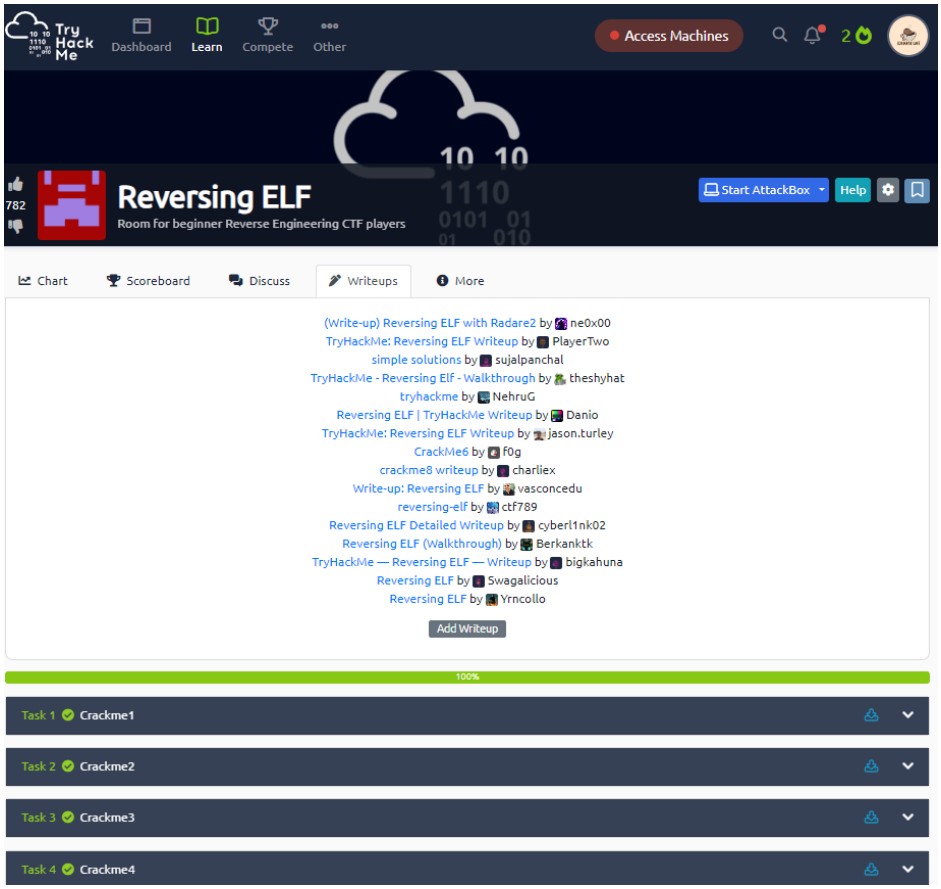
For *Task 4* this one was tricky, but found that if I open the defined strings window found on the toolbar I was able to find the flag stack. We can see in the parameters that in hex it calls ID 0x110 which is 272 in decimal.

```
16  local_c = FindResourceA((HMODULE)0x0,"rc.rc",(LPCSTR)0x6);
17  local_8 = 0x110;
18  LoadStringA((HINSTANCE)0x0,0x110,&local_4a4,0x3ff);
19  local_a0 = MD5::digestString(local_9c,&local_4a4);
```

Now we can hop over to defined strings and click one of the flags in the stack. Then we look at the Rsrc String ID and look for ID 272. There we will find the flag in plaintext.

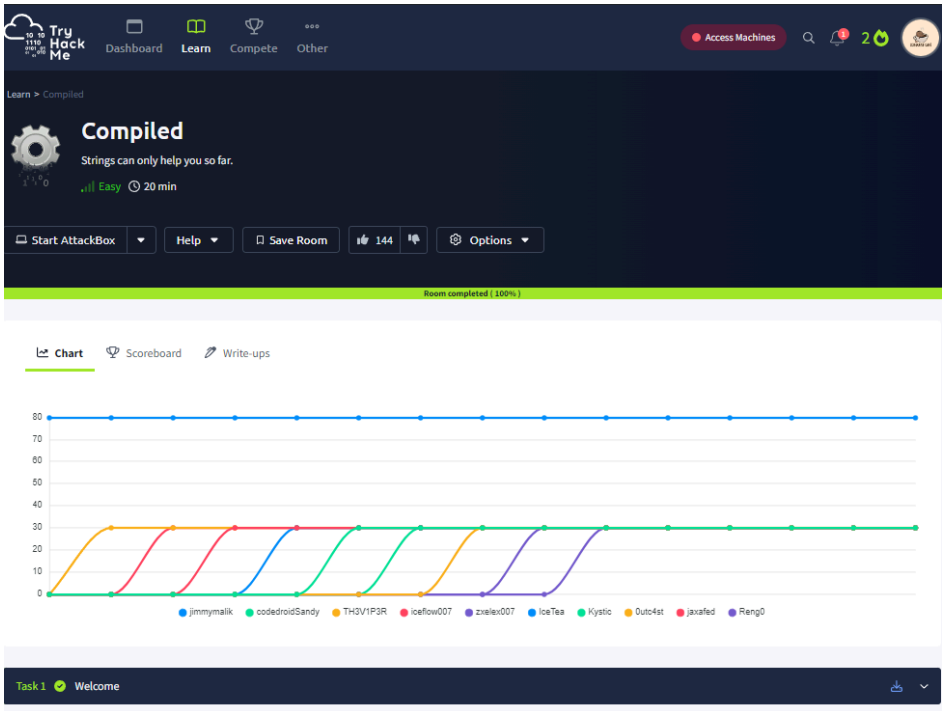
```
:_StringTable_12_409                          XREF[1]:     entry:004022ff(*)
 p_unicode   u"FLAG{RESOURCES-ARE-POPULAR-FOR-MALWARE}"      Rsrc String ID 272
```

Task 2 Reverse ELF files;

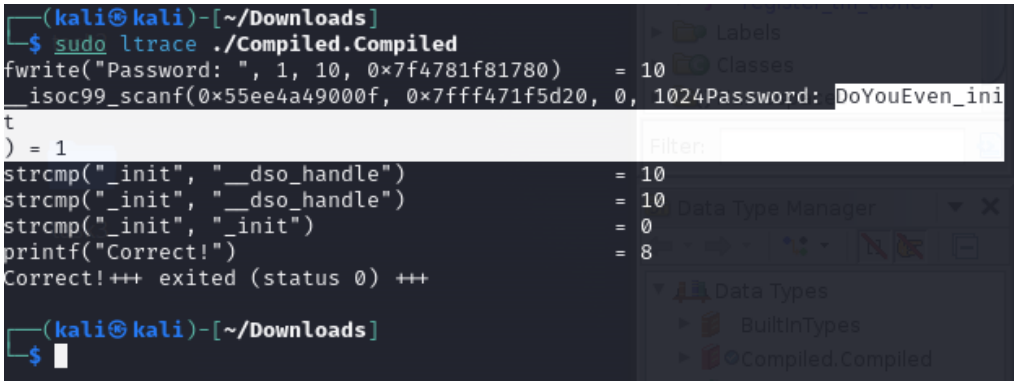


Remember how to use chmod, strings, & cyberchef. Crackme3 is encoded in base64. Then between using ghidra and gdb I was able to get the addresses and the registers to find all the information I needed to complete the rest of the flags.

Task 3 Compiled;

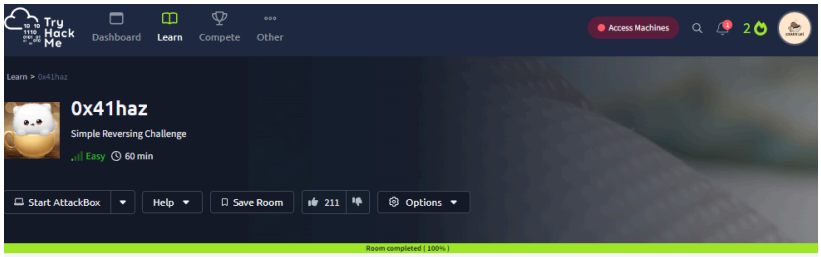


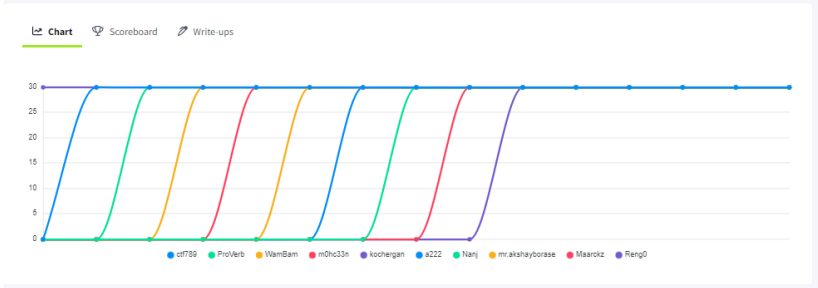
This one was cool because we can use some exploiting to get the program to work. I do this by reading the main function to see that line 9 scans for DoYouEven%s and then takes the input followed by CTF. It later compares the input to "_init" so I ran ltrace on the program to pass the correct value.

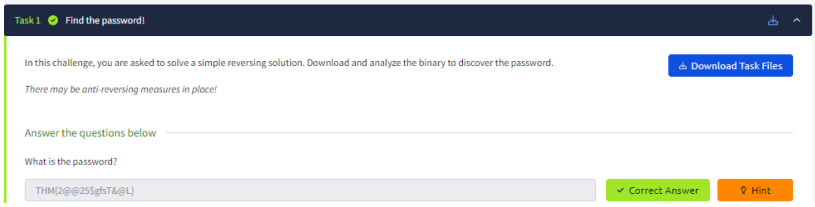


I tried putting this value into the box and it worked

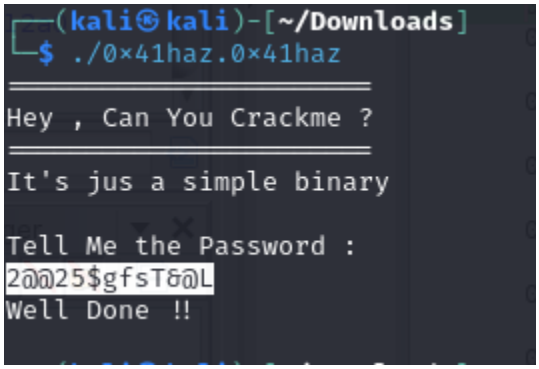
Task 4 0x41haz;



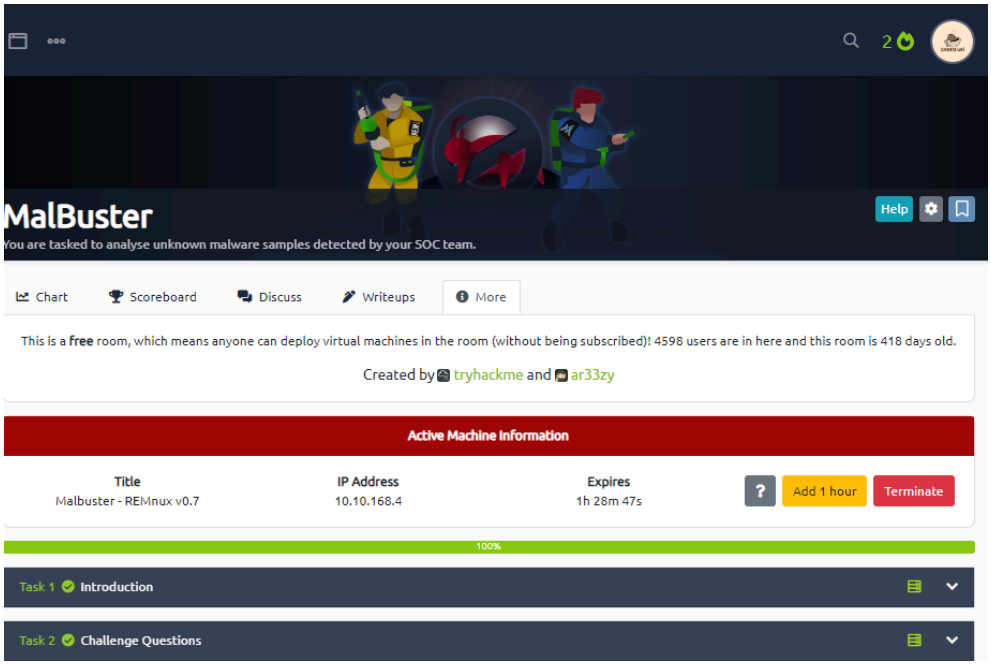




I didn't know how to disassemble a true executable so I had to look at some help where it was recommended I go to hexed.it and change the 6th byte from 02 to 01. Then I downloaded that file to be able to open it in ghidra. It had a starting point function called entry which was tricky, but I was able to set a breakpoint in gdb to run through the program. I found some hex values in FUN_00101165 which are in Little Endian form. After an online converter I was able to input the password and we are in;

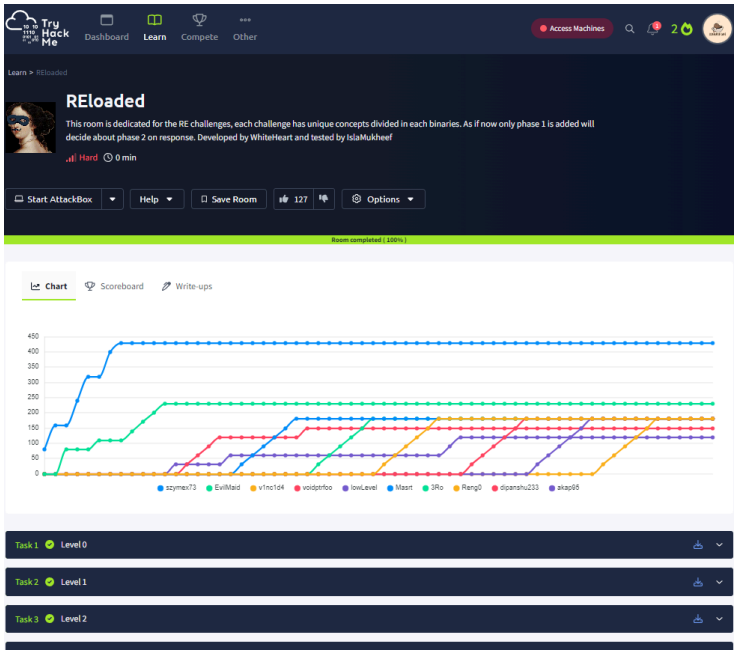


Task 5 malbuster;



This one made me open the webVM, so I chose REMnux because I prefer Kali Linux to Windows for reverse engineering. I then went through all the hash questions using the import file and using the MD5 hash that is displayed and threw it into VirusTotal. I got taken to the wrong website to find out it's 'bazaar'.abuse.ch not abuse.ch. I eventually found it and was able to find the correct signature for MalBuster_3

Bonus Reloaded;



For the first couple of tasks I was able to get away with parsing the strings of the executable. Then I had to use CyberChef to decode the if statement hex code. Found the decimal value of that and got the flag. Ghidra allows me to export patched files if you change the jnz instruction to a jz instruction and run the program. I can get the program to think I gave it the right password, giving me the flag. I learned how to use Immunity Debugger, I first used ghidra to layout the functions and addresses, then I used immunity debugger in place of gdb allowing me to look into the registers. The encrypted one was tricky until I found the MOV instruction through the FUN_004014e6. There I was able to find the flag in the EAX.