Challenge 2: Huge

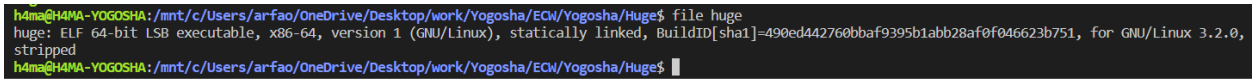
Description:
This challenge will consist of a large binary that will be hard to decompile so the player must find a way to get the byte codes and then generate the flag

Description:
Wow this is huge can you find my Flag? It got lost in there.

Flag:
FLAG{THISISHUGEOHHHHHHHHHHHH}

Solution:

So in this challenge, we have an ELF 64-bit binary



We start by trying the static analysis with IDA but when loading it to IDA it crashes.

And even using gdb we can't have any info about the functions of anything so we use pyelftools to extract the assembly

Script:

```
from elftools.elf.elffile import ELFFile
from capstone import *


file = open("code.txt", "a")
with open('./huge', 'rb') as f:
    elf = ELFFile(f)
    code = elf.get_section_by_name('.text')
    ops = code.data()
    addr = code['sh_addr']
    md = Cs(CS_ARCH_X86, CS_MODE_64)
    for i in md.disasm(ops, addr):
        file.write(f'0x{i.address:x}:\t{i.mnemonic}\t{i.op_str}\n')
```

So now that we have the code we start the analysis after some while we can find that there is a code pattern that takes some value from the memory and after doing an operation it compares it with a number so knowing that the program takes input from a file we can say that this is the check.

```
0x401d6d:   mov qword ptr [rbp - 8], rdi
0x401d71:   mov rax, qword ptr [rbp - 8]
0x401d75:   movzx   eax, byte ptr [rax]
0x401d78:   cmp al, 0x4c
0x401d7a:   jne 0x5b7b98
0x401d80:   mov rax, qword ptr [rbp - 8]
0x401d84:   add rax, 1
0x401d88:   movzx   eax, byte ptr [rax]
0x401d8b:   cmp al, 0x6f
0x401d8d:   jne 0x5b7b98
0x401d93:   mov rax, qword ptr [rbp - 8]
0x401d97:   add rax, 2
0x401d9b:   movzx   eax, byte ptr [rax]
0x401d9e:   cmp al, 0x72
0x401da0:   jne 0x5b7b98
0x401da6:   mov rax, qword ptr [rbp - 8]
0x401daa:   add rax, 3
0x401dae:   movzx   eax, byte ptr [rax]
0x401db1:   cmp al, 0x65
0x401db3:   jne 0x5b7b98
```

Now as we see that we only need the values so we can build the correct input, we use the following script to extract them:

```
file = open("code.txt", "r")
lines=file.readlines()
f = open("values.txt", "a")
i=768
while i<=427874:
```

```
    f.write(","+lines[i][18:])
    i+=5
```

Now we have the values we just build the input

```
file = open("code.txt", "r")
lines=file.readlines()
f = open("values.txt", "a")
#print(int(lines[768][18:],16))
i=768
while i<=427874:
    f.write(chr(int(lines[i][18:],16)))
    i+=5
```