

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{THISISHUGEOHHHHHHHHHHHHHH}

Solution:

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

```
h4ma@H4MA-YOGOSHA: /mnt/c/Users/ar-fao/OneDrive/Desktop/work/Yogosha/ECW/Yogosha/Huge$ file huge
huge: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, BuildID[sha1]=490ed442760bbaf9395b1abb28af0f046623b751, for GNU/Linux 3.2.0,
stripped
h4ma@H4MA-YOGOSHA: /mnt/c/Users/ar-fao/OneDrive/Desktop/work/Yogosha/ECW/Yogosha/Huge$
```

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}: {i.mnemonic} {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
```

```
consectetur lorem donec massa.Netus et malesuada fames ac turpis egestas. Risus  
utpat commodo sed. Purus sit amet luctus venenatis lectus magna. Elit sed vulput  
amet nisl purus in. Mauris vitae ultricies leo integer malesuada nunc. Habitant  
sa enim nec dui. Tincidunt eget nullam non nisi. Eget sit amet tellus cras adipi  
s sed felis eget. FLAG{THISISHUGE0HHHHHHHHHHHH} Lorem ipsum dolor sit amet, cons  
ore et dolore magna aliqua. Cursus risus at ultrices mi tempus imperdiet nulla m  
u. Eu turpis egestas pretium aenean pharetra magna. Bibendum enim facilisis grav  
blandit. Tellus rutrum tellus pellentesque eu tincidunt. Auctor elit sed vulput  
tium vulputate sapien nec sagittis aliquam malesuada bibendum arcu vitae. Viverr  
ectus et netus et malesuada fames. Diam donec adipiscing tristique risus nec fe  
tibulum. Velit scelerisque in dictum non consectetur a. Egestas sed tempus urna  
c. Pharetra convallis posuere morbi leo urna molestie at. Mauris a diam maecenas
```