CS HW0

to bf or not to bf

From the files given, we can reasonably guess that there were two original images. And after running the script, two images became encrypted to the ones we have.

The encrypt method is simply xor a random number to every pixel of each image.

Since xor a number to a another number twice is equal to do nothing `a^c^c=a` , therefore there is an observation that `(a^c)^(b^c)=a^b` .

So by the instinct, I simply xor two images's every pixel together and get the flag.

XAYB

I use ghidra to disassemble the the binary to something looks like c code. At the last part of the function `game_logic` , it first xor a number to every element of an array, and print it out finally. So we can guess that it might be the flag we want.

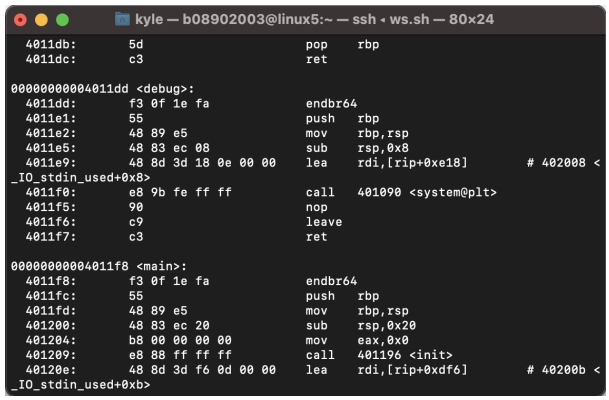
From the clue, I trace back to the origin of that mystery array, which is come from `main` function.

So I write a program to combine these strange variables to an array, and xor 0xf2 to each byte of them as the last part of `game_logic` doing and get the flag finally.

Please check out my code for detail.

Arch Check

First, I use `objdump -D -M intel -j .text arch_check` to disassemble the binary.


kyle — b08902003@linux5:~ — ssh ‹ ws.sh — 80×24
4011db: 5d pop rbp
4011dc: c3 ret
00000000004011dd <debug>:
4011dd: f3 0f 1e fa endbr64
4011e1: 55 push rbp
4011e2: 48 89 e5 mov rbp,rsp
4011e5: 48 83 ec 08 sub rsp,0x8
4011e9: 48 8d 3d 18 0e 00 00 lea rdi,[rip+0xe18] # 402008 <
_IO_stdin_used+0x8>
4011f0: e8 9b fe ff ff call 401090 <system@plt>
4011f5: 90 nop
4011f6: c9 leave
4011f7: c3 ret
00000000004011f8 <main>:
4011f8: f3 0f 1e fa endbr64
4011fc: 55 push rbp
4011fd: 48 89 e5 mov rbp,rsp
401200: 48 83 ec 20 sub rsp,0x20
401204: b8 00 00 00 00 mov eax,0x0
401209: e8 88 ff ff ff call 401196 <init>
40120e: 48 8d 3d f6 0d 00 00 lea rdi,[rip+0xdf6] # 40200b <
_IO_stdin_used+0xb>">

We can find that there is a function called `debug` at the address 0x4011dd.

And in the `main` function, we can find a buffer with size 0x20 bytes before calling scanf, so there might be a chance that we can send 0x20+0x8 bytes data to scanf to cause a buffer overflow and replace the return address to what we want.

Therefore, we can make the process jump to the function `debug` which calls `shell`.

After getting control of shell, we can find the flag in the home directory.

Please check out my code for detail.

text2emoji

From the source code, we can find out that we can only access to `/public_api`, which sends another request to the private api `/emoji/:text` and get the emoji back.

But the interesting point is that there is another private api `/looksLikeFlag` which can tell whether the flag contains the query string, therefore the main target is to get access to this api somehow.

By the part of redirecting the public api to the private api, there is a vulnerability that we can use directory traversal attack to change the url.

So we can access to `/looksLikeFlag` by sending request to `/public_api` with the following request body:

```
{'text': "%2e%2e/looksLikeFlag?flag={flag}"}
```

Finally, we can write a simple script to find out the complete flag by querying the api.