R09921A10 鄭翔予
CTF Account: stanleymusic

1. to bf or not to bf
   - Description
     - Given two encrypted images and the encryption algorithm code, try to get the flag
   - Solution
     - The two images are encrypted using the same key
     - The encryption algorithm is simply XOR on each byte
     - Thus the encrypted images we get are in the format of
       - flag ^ key
       - golem ^ key

     - flag ^ key ^ golem ^ key = flag ^ golem
     - Since the images are just grayscale, eliminating the key part might be clear enough to know the original flag and golem
   - Flag
     - FLAG{Do_not_teLL_Golem_i_use_his_photo}
2. XAYB
   - Description
     - Given a binary file, try to get the flag
   - Solution
     - Use IDA Pro to view the source code of the given binary file
     - The interesting part lies in the "game_logic" function
     - Try to make the code execute the "BINGO!" part and the code will output the flag to us. Furthermore, the process of generation of the flag isn't affected by our input, which means that the input doesn't matter as long as the condition is met

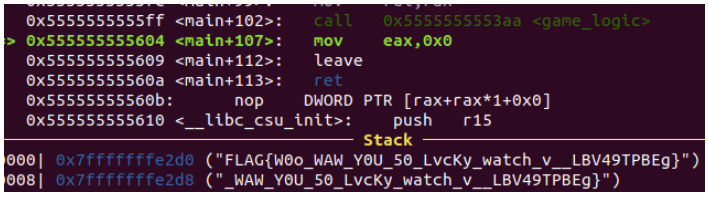
     ```
     if ( v9 == 5 )
     {
       puts("BINGO!");
       for ( l = 0; l <= 45; ++l )
         a1[l] ^= 0xF2u;
       return puts(a1);
     }
     ```

     - The condition is v9==5, we can modify it during execution using gdb
     - Set breakpoint at condition and use gdb instruction("set $addr = $value") to modify the address value

     ```
     Breakpoint 2, 0x0000555555555518 i
     gdb-peda$ x/s $rbp-0xc
     0x7fffffffe2b4: "\003"
     gdb-peda$ set *0x7fffffffe2b4 = 5
     ```

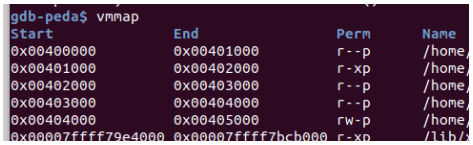
     - Use gdb instruction ("fin") to step out of the function, the flag is stored on the top of the stack

- Flag



    - FLAG{W0o_WAW_Y0U_50_LvcKy_watch_v__LBV49TPBEg}

3. Arch Check
    - Special thanks to Yi-Hsien Chen for hinting me the format_s is not successfully written to the memory (Due to the bad character issue, which I found it after searching online)
    - Description
        - Given a binary file, try to get the flag
    - Solution
        - Use r2 to analyze the binary file, we can find some interesting functions including system, puts, scanf, and main
        - The main function is simple. It includes three puts and one scanf function, and the scanf doesn't specify the input length, which means there is a buffer overflow issue
        - To get the shell, we need to try to execute "system('/bin/sh')", unfortunately, there aren't any strings including '/bin/sh', but with the help of scanf we can first write '/bin/sh' to an writable space and then call "system" with this address as a parameter
            - Use gdb instruction ("vmmap") to find writable space



        - The ROP chain looks like following
            - pop $rdi, %s
            - pop $rsi, binsh_addr
            - scanf
            - pop $rdi, binsh_addr
            - ret
            - system
        - The ret before the system is only used to make sure the instruction is 16-byte alignment, otherwise, the binary file will refuse to execute the ROP chain
        - Use ROPgadget to help find useful pop and ret gadgets
        - Avoid writing bad characters to address such as 0x0D('\r'), 0x0A('\n'), 0x20(' ')

        - In this case, one of the "%s" is at the address including 0x20
- Flag
    - FLAG{d1d_y0u_ju5t_s4y_w1nd0w5?}

4. text2emoji
    - Description
        - Given the source code of the website, try to get the flag
    - Solution
        - Observe the source code, the URL part is controllable

```
const url = `http://127.0.0.1:7414/api/v1/emoji/${text}`;
http.get(url, result => {
```

        - Moreover, there is an API at '/looksLikeFlag' which can help us determine whether our input of flag parameter is identical to the flag

```
});
apiRouter.use('/looksLikeFlag', (req, res, next) => {
    assert(FLAG.match(/^FLAG{[a-z0-9_]+}$/));

    res.send({ looksLikeFlag: FLAG.includes(req.query.flag) });
});
```

        - To access '/looksLikeFlag' we need to do path traversal from '/emoji/${text}' to '/looksLikeFlag', however some special character is blocked including '.'

```
if (!text.match(/^\S+$/) || text.includes(".")) {
    response.send({ error: "Bad parameter" });
    return;
}
```

        - There are still other ways to bypass character blocking, which can be found on OWASP. Here, the encoding variation works ("%2e%2e/")

```
%2e%2e/looksLikeFlag?flag=FLAG{
```

        - Now, we can access the '/looksLikeFlag' API, the remaining part is easy, just need to brute force the FLAG
            - Javascript "includes" function will return TRUE whenever the request are all included
            - Start by giving 'FLAG{' + x, where x is a variable in "abcdefghijklmnopqrstuvwxyz0123456789_}", and if the response is {"looksLikeFlag":true} then we get the right character, brute force until '}' is matched
            - I use Burp Suite and its plugin Turbo Intruder to do the brute force
- Flag
    - FLAG{3asy_p4th_tr4vers4l}