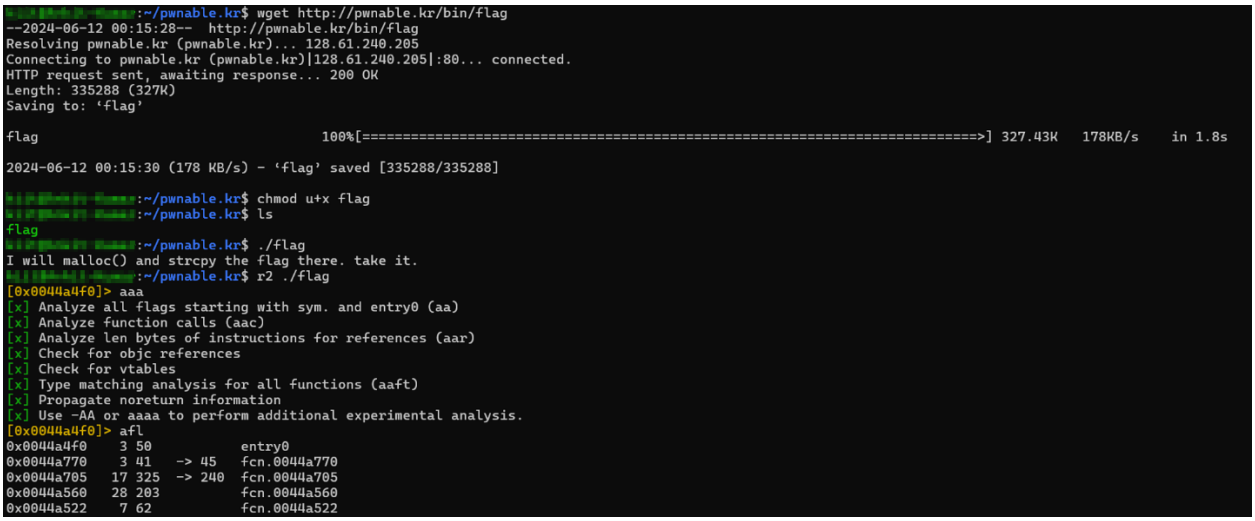
flag

This is a writeup of reverse engineering challenge from pwnable.kr called “flag”.

I used Radare2 to solve this challenge, so let’s get started.

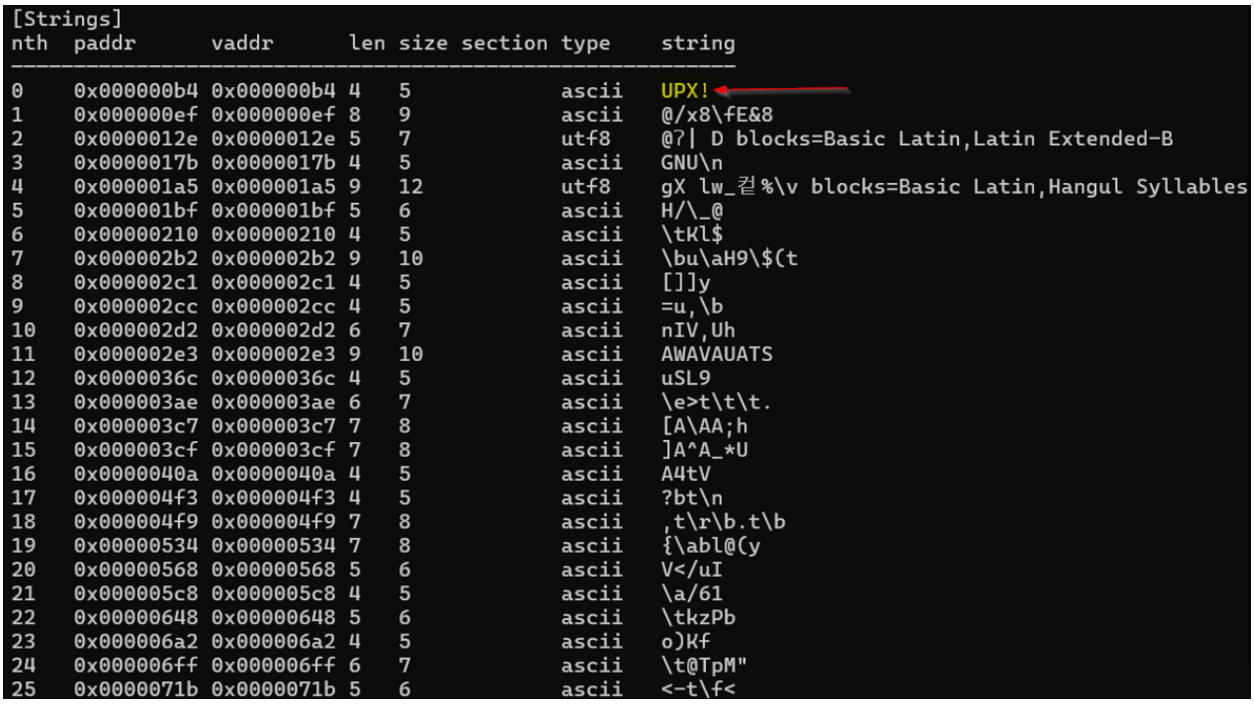
```
r2 ./flag    //Let's go through the file first
DON'T forget to change the permissions of the file using chmod.
```

Now, let’s analyze the file and see what functions are there,



Also let’s see the strings that this binary file contains:

```
azz //This is the command to view strings
or
azz ~.. //This basically using less to that command
```

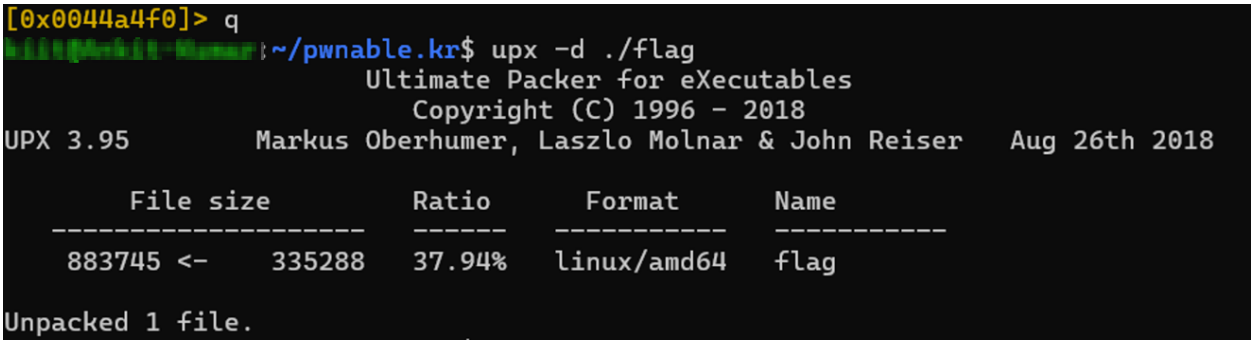


```
[Strings]
nth  paddr      vaddr      len size section type    string
-------------------------------------------------------------
0    0x000000b4 0x000000b4 4   5           ascii   UPX!
1    0x000000ef 0x000000ef 8   9           ascii   @/x8\fE&8
2    0x0000012e 0x0000012e 5   7           utf8    @?| D blocks=Basic Latin,Latin Extended-B
3    0x0000017b 0x0000017b 4   5           ascii   GNU\n
4    0x000001a5 0x000001a5 9   12          utf8    gX lw_컫%\v blocks=Basic Latin,Hangul Syllables
5    0x000001bf 0x000001bf 5   6           ascii   H/\_@
6    0x00000210 0x00000210 4   5           ascii   \tKl$
7    0x000002b2 0x000002b2 9   10          ascii   \bu\aH9\$(t
8    0x000002c1 0x000002c1 4   5           ascii   []]y
9    0x000002cc 0x000002cc 4   5           ascii   =u,\b
10   0x000002d2 0x000002d2 6   7           ascii   nIV,Uh
11   0x000002e3 0x000002e3 9   10          ascii   AWAVAUATS
12   0x0000036c 0x0000036c 4   5           ascii   uSL9
13   0x000003ae 0x000003ae 6   7           ascii   \e>t\t\t.
14   0x000003c7 0x000003c7 7   8           ascii   [A\AA;h
15   0x000003cf 0x000003cf 7   8           ascii   ]A^A_*U
16   0x0000040a 0x0000040a 4   5           ascii   A4tV
17   0x000004f3 0x000004f3 4   5           ascii   ?bt\n
18   0x000004f9 0x000004f9 7   8           ascii   ,t\r\b.t\b
19   0x00000534 0x00000534 7   8           ascii   {\abl@(y
20   0x00000568 0x00000568 5   6           ascii   V</uI
21   0x000005c8 0x000005c8 4   5           ascii   \a/61
22   0x00000648 0x00000648 5   6           ascii   \tkzPb
23   0x000006a2 0x000006a2 4   5           ascii   o)Kf
24   0x000006ff 0x000006ff 6   7           ascii   \t@TpM"
25   0x0000071b 0x0000071b 5   6           ascii   <-t\f<
```

After running strings, I found this UPX! at the top. Upon further googling, I found that this a technique that can be used to run the program while being compressed and was often used in malwares.

So, lets quit the radare and quickly decompress it.

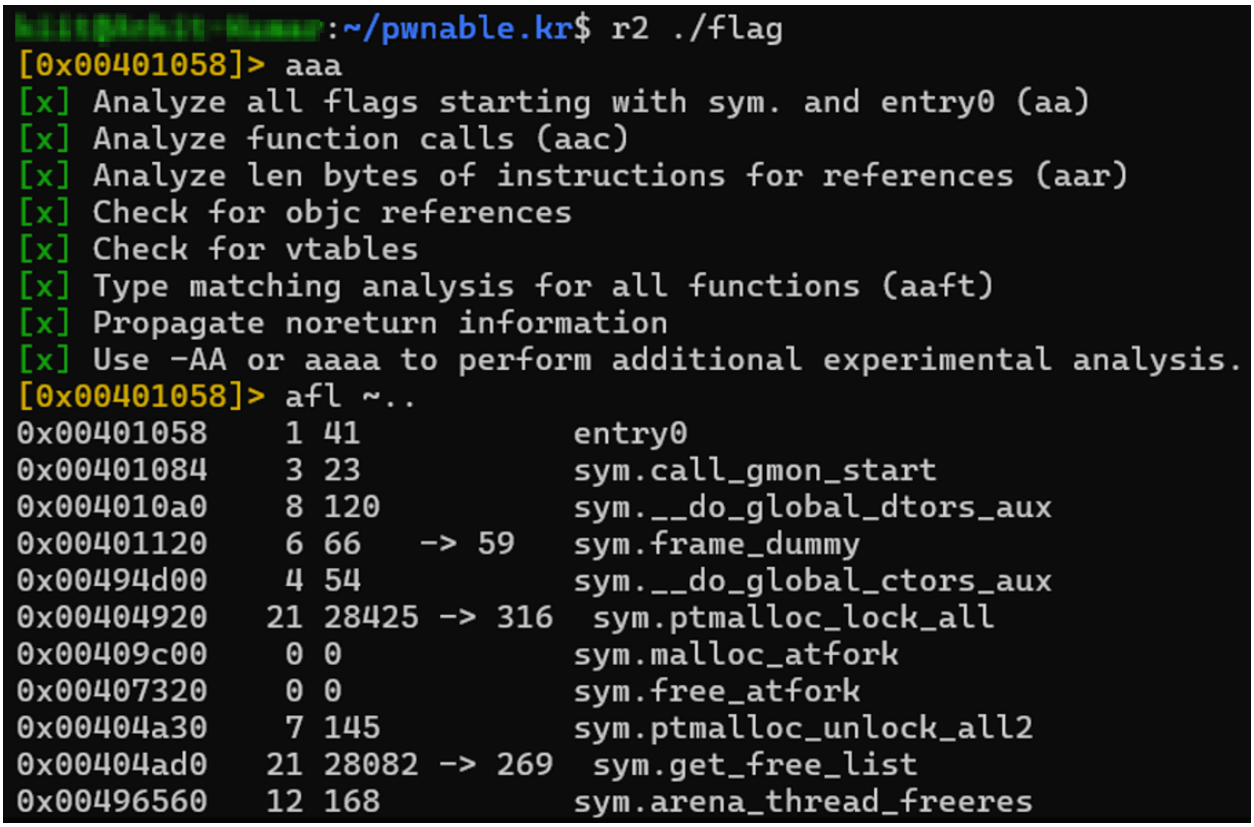
```
sudo apt install upx-ucl   //Installing the upx

upx -d ./flag              //Decompressing the file
```



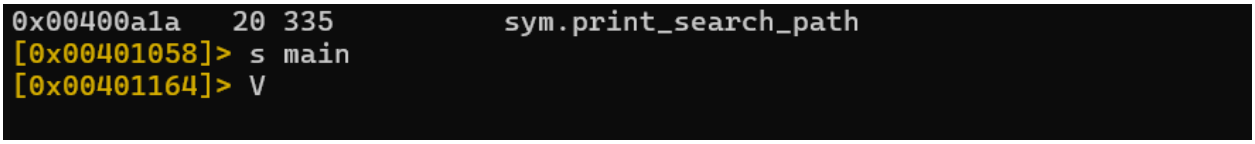
```
[0x0044a4f0]> q
[redacted]:~/pwnable.kr$ upx -d ./flag
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2018
UPX 3.95        Markus Oberhumer, Laszlo Molnar & John Reiser   Aug 26th 2018

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    883745 <-    335288   37.94%   linux/amd64   flag

Unpacked 1 file.
```

It automatically overwrites the file after decompressing it.

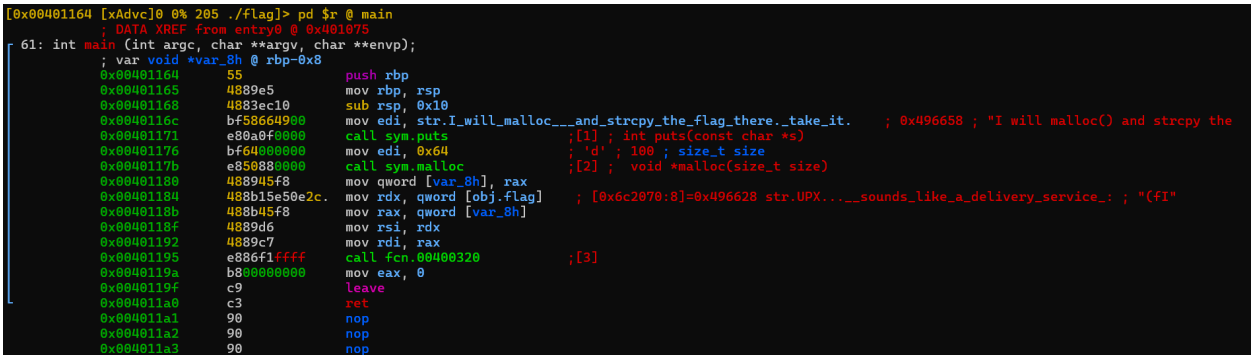
Now lets run the program again using radare2.



Now let's seek to main function and view it.




```
Hit p till you reach the decompiled view of the binary file.
You can also hit P if you wanna go backwards.
```



Here we can see that there are three functions in total, one is puts, the other one is malloc and according to the author it says there's also strcpy function so the last one must be it.

To solve this and get the flag, we will go to the address after the strcpy function call, and print the flag from var_8h.

To do that let's enter debug mode in radare2.

Quit the program, use 'q' to do that.

And re run the program in debug mode.

```
r2 -d ./flag
```

```
[redacted]:~/pwnable.kr$ r2 -d ./flag
Process with PID 2017 started...
= attach 2017 2017
bin.baddr 0x00400000
Using 0x400000
asm.bits 64
[0x00401058]> aaa
[Invalid address from 0x00494113ith sym. and entry0 (aa)
Invalid address from 0x00494132
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Check for objc references
[x] Check for vtables
[TOFIX: aaft can't run in debugger mode.ions (aaft)
[x] Type matching analysis for all functions (aaft)
[x] Propagate noreturn information
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x00401058]> s main
[0x00401164]>
```

Analyze the flag, wait a few minutes for that to analyze the binary file.

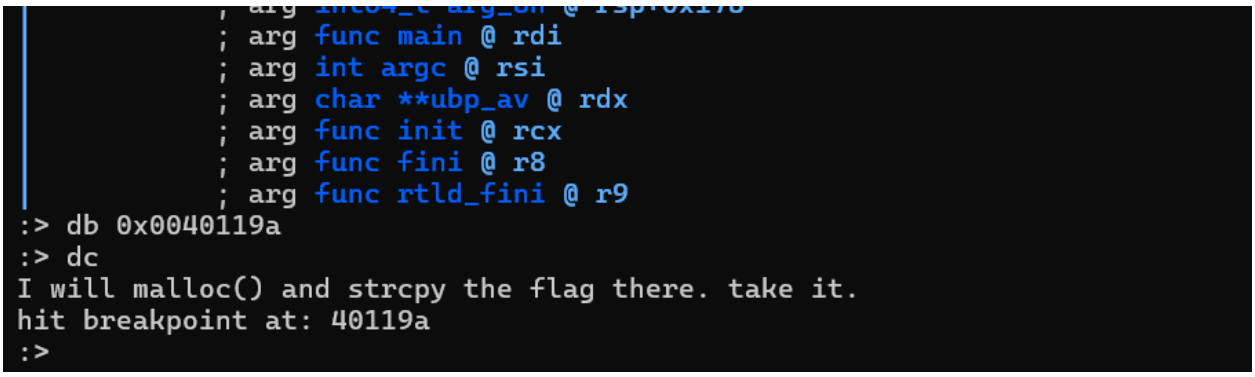
Then seek to main function. Hit ‘V’ to enter visual mode.

Now navigate to the debug view by pressing ‘p’ and going forward.

```
[0x00401164 [xAdvc]0 0% 205 ./flag]> pd $r @ main
            ; DATA XREF from entry0 @ 0x401075
┌ 61: int main (int argc, char **argv, char **envp);
│           ; var void *var_8h @ rbp-0x8
│           0x00401164      55              push rbp
│           0x00401165      4889e5          mov rbp, rsp
│           0x00401168      4883ec10        sub rsp, 0x10
│           0x0040116c      bf58664900      mov edi, str.I_will_malloc___and_strcpy_the_flag_there._take_
│           0x00401171      e80a0f0000      call sym.puts               ;[1] ; int puts(const char *s)
│           0x00401176      bf64000000      mov edi, 0x64               ; 'd' ; 100 ; size_t size
│           0x0040117b      e850880000      call sym.malloc             ;[2] ;  void *malloc(size_t size)
│           0x00401180      4889f5f8        mov qword [var_8h], rax
│           0x00401184      488b15e50e2c.   mov rdx, qword [obj.flag]    ; [0x6c2070:8]=0x496628 str.UPX.
│           0x0040118b      488b45f8        mov rax, qword [var_8h]
[0x00000000 [xAdvc]0 0% 16384 ./flag]> pd $r @ sym.__libc_tsd_LOCALE
            ;-- section.:
            ;-- section..comment:
```

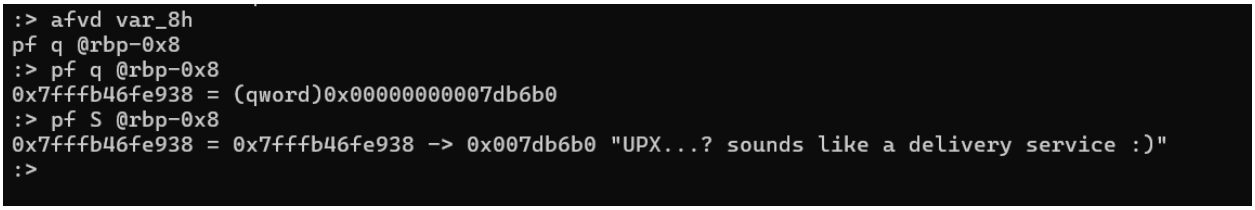
Now, set a breakpoint at the address ‘0x0040119a’ just after the function call. To do that hit ‘:’ semicolon. Then,

```
db 0x0040119a      //set a breakpoint at this address
dc                 //countinue execution
```



Now lets read the address where the flag is:

```
afvd var_8h      //afvd is used to show the value of args/local

pf S @rbp-0x8   //printf string at rbp-0x8
```




```
And we got the flag:

“UPX...? sounds like a delivery service :)”
```