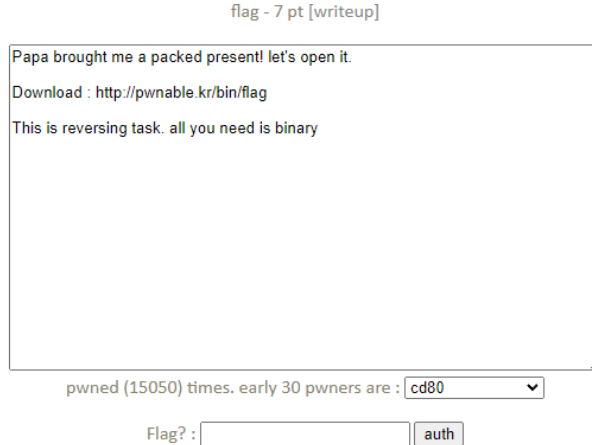


Pwnable.kr

- flag -

Download : <http://pwnable.kr/bin/flag>

0. 문제 살펴보기



문제에서는 bof 때와 같이 파일을 다운로드하여 살펴보는 것 같다.
원격 접속 주소가 존재하지 않으니, 해당 파일에 flag가 숨겨져 있는 것으로 생각된다.
이 문제는 리버싱에 관한 것이라고 하며 바이너리가 필요하다고 한다.

1. 파일 다운로드 및 살펴보기

```
(kali@kali)-[~]
└─$ wget http://pwnable.kr/bin/flag
--2024-01-31 14:24:51-- http://pwnable.kr/bin/flag
Resolving pwnable.kr (pwnable.kr)... 128.61.240.205
Connecting to pwnable.kr (pwnable.kr)|128.61.240.205|:80 ... connected.
HTTP request sent, awaiting response... 200 OK
Length: 335288 (327K)
Saving to: 'flag'

flag 100%[=====] 327.43K 429KB/s in 0.8s
2024-01-31 14:24:52 (429 KB/s) - 'flag' saved [335288/335288]
```

```
(kali@kali)-[~]
└─$ file flag
flag: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), statically linked, for GNU/Linux 2.6.24, BuildID[sha1]=96ec4cc272aeb383bd9ed26c0d4ac0eb5db41b16, not stripped
```

wget 명령을 이용하여 파일을 다운로드하고 flag 파일을 살펴보았다.
file 명령을 이용하여 해당 파일이 무엇인지 확인해보았더니 실행 파일인 것을 확인할 수 있었다.

```
(kali@kali)-[~]
└─$ ./flag
I will malloc() and strcpy the flag there. take it.
```

flag 파일을 실행해보니 malloc과 strcpy를 했고 flag는 그곳에 있다고 한다.

2. gdb

```
(kali@kali)-[~]
└─$ gdb flag
GNU gdb (Debian 13.2-1) 13.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.  "The quieter you become, the more you are able to hear"
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word" ...
Reading symbols from flag...
(gdb) disas main
No debugging symbols found in flag
No symbol table is loaded. Use the "file" command.
```

디버깅 정보를 확인하기 위해 gdb를 이용하여 flag 파일을 열어 main 함수의 어셈블리어를 보려고 하였으나, 심볼이 존재하지 않다는 확인이 불가능한 것을 볼 수 있다.

3. 바이너리 확인

```
(kali@kali)-[~]
$ xxd flag
00000000: 7f45 4c46 0201 0103 0000 0000 0000 0000  .ELF.....
00000010: 0700 3e00 0100 0000 f0a4 4400 0000 0000  ..>...D....
00000020: 4000 0000 0000 0000 0000 0000 0000 0000  @.....
00000030: 0000 0000 4000 3800 0200 4000 0000 0000  ....@.8...@....
00000040: 0100 0000 0000 0000 0000 0000 0000 0000  .a.....
00000050: 0000 4000 0000 0000 0000 4000 0000 0000  ..a.....
00000060: 0xad 0400 0000 0000 0xad 0400 0000 0000  ...
00000070: 0000 2000 0000 0000 0100 0000 0000 0000  ...
00000080: d862 0c00 0000 0000 d862 6c00 0000 0000  .b.....bl....
00000090: d862 6c00 0000 0000 0000 0000 0000 0000  .bl.....
000000a0: 0000 0000 0000 0000 0000 2000 0000 0000  .....
000000b0: fcac e8a1 5550 5821 1c08 0d16 0000 0000  ...UPX!....
000000c0: 217c 0d00 217c 0d00 9001 0000 9000 0000  !|..|!....
000000d0: 0000 0000 f7fb 93ff 7f45 4c46 0201 0103  ....ELF....
```

문제에서 우리는 binary가 필요하다고 하였다. 그렇기 때문에 xxd를 이용하여 flag 파일을 binary로 확인하였다. 그러자 UPX라는 한 단어가 눈에 띈다. 이 파일은 현재 압축되어 있어 gdb가 정상적으로 작동하지 않은 것으로 생각된다.

※ UPX

수많은 파일 포맷을 지원하는 오픈소스 실행 파일 압축 프로그램이다.

압축은 UCL이라는 이름의 데이터 압축 알고리즘을 사용한다.

압축 해제에는 “in-place technic”과 “임시 파일로의 해제” 이렇게 두 가지의 메커니즘을 지원한다.

4. 압축 해제 및 gdb

```
(kali@kali)-[~]
$ upx -d flag
Ultimate Packer for eXecutables
Copyright (C) 1996 - 2020
UPX 3.96      Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd 2020

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

Unpacked 1 file.
```

flag 파일이 upx로 압축되었다는 것을 알았으니 해제할 차례이다.

upx의 -d 옵션을 이용하여 flag 파일을 압축 해제하였다.

```
(kali@kali)-[~]
$ gdb flag
(gdb) disas main
Dump of assembler code for function main:
0x0000000000401164 <+0>: push    %rbp
0x0000000000401165 <+1>: mov     %rsp,%rbp
0x0000000000401168 <+4>: sub     $0x10,%rsp
0x000000000040116c <+8>: mov     $0x496658,%edi
0x0000000000401171 <+13>: call    0x402080 <puts@plt>
0x0000000000401176 <+18>: mov     $0x64,%edi
0x000000000040117b <+23>: call    0x4099d0 <malloc@plt>
0x0000000000401180 <+28>: mov     %rax,-0x8(%rbp)
0x0000000000401184 <+32>: mov     0x2c0ee5(%rip),%rdx      # 0x6c2070 <flag>
0x000000000040118b <+39>: mov     -0x8(%rbp),%rax
0x000000000040118f <+43>: mov     %rdx,%rcl
0x0000000000401192 <+46>: mov     %rax,%rdi
0x0000000000401195 <+49>: call    0x400320
0x000000000040119a <+54>: mov     $0x0,%eax
0x000000000040119f <+59>: leave   %eax
0x00000000004011a0 <+60>: ret
```

이후, gdb로 flag 파일을 열자 정상적으로 작동하는 것을 볼 수 있다.

main의 어셈블리어를 확인하자 한 가지 눈에 띄는 문구가 발견되었다.

“0x6c2070 <flag>” 혹시 몰라 해당 위치의 값을 확인해보았다.

```
(gdb) x/1s *0x6c2070
0x496658: " "
```

그랬더니 어떠한 문장이 등장하였다. 우리가 다른 단계에서 보았던 문장과 같이 어떠한 대화 방식의 문장이어서 사이트에서 flag로 입력하였더니 올바른 문장이었다.