sefi-roee / CTFs-Writeups



136 lines (92 sloc) 3.85 KB

Problem

Can you deal with the Duck Web? Get us the flag from this program. You can also find the program in /problems/quackme_3_9a15a74731538ce2076cd6590cf9e6ca.

Hints:

Objdump or something similar is probably a good place to start.

Solution:

First lets download the file and try to execute it

```
wget https://2018shell1.picoctf.com/static/1d21f78fd2b82ebff2ad54a8b09081c8/main
chmod +x ./main
./main

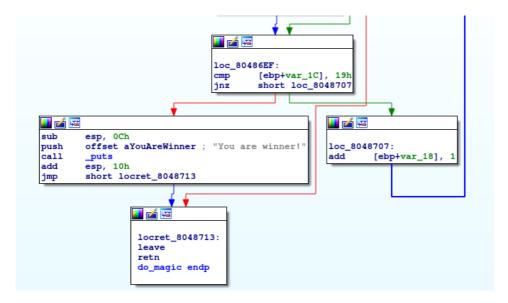
You have now entered the Duck Web, and you're in for a honkin' good time.
Can you figure out my trick?
<INPUT>
That's all folks.
```

Lets try to understand whats going on there, we can use IDA for disassembly (or even objdump/gdb)

```
<u></u>
; Attributes: bp-based frame fuzzy-sp
; int __cdecl main(int argc, const char **argv, const char **envp) public main main proc near
var_4= dword ptr -4
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h
lea
          ecx, [esp+4]
esp, 0FFFFFF0h
dword ptr [ecx-4]
and
push
push
           ebp
           ebp, esp
push
           ecx
sub
           esp, 4
           eax, ds:stdout@@GLIBC_2_0
mov
push
push
push
push
call
           eax
_setvbuf
           esp, 10h
esp, 0Ch
add
sub
push
           offset aYouHaveNowEnte ; "You have now entered the Duck Web, and "...
           _puts
esp, 10h
do_magic
call
add
call
           esp, 0Ch
offset aThatSAllFolks ; "That's all folks."
sub
push
call
           _puts
          esp, 10h
eax, 0
ecx, [ebp+var_4]
add
mov
mov
leave
lea
          esp, [ecx-4]
retn
main endp
```

And do_magic():

```
; Attributes: bp-based frame
                                 public do_magic
do_magic proc near
                                  var_1D= byte ptr -1Dh
                                 var_1C= dword ptr -1Ch
var_18= dword ptr -18h
                                 var_14= dword ptr -14h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
                                 push
mov
                                             ebp
                                             ebp, esp
                                            esp, 28h
read_input
[ebp+var_14], eax
                                  sub
                                  call
                                  mov
                                  sub
                                             esp, 0Ch
                                            [ebp+var_14]
_strlen
                                 push
call
                                  add
                                             esp, 10h
                                            [ebp+var_10], eax
eax, [ebp+var_10]
eax, 1
                                  mov
                                  add
                                             esp, OCh
                                  sub
                                  push
                                             eax
                                            _malloc
esp, 10h
[ebp+var_C], eax
[ebp+var_C], 0
short loc_8048696
                                  call
                                  add
                                  mov
                                  cmp
                                  jnz
  <u></u>
             esp, 0Ch
offset aMallocReturned ; "malloc() returned NULL. Out of Memory\n
  sub
  push
             _puts
esp, 10h
  call
  add
  sub
             esp, 0Ch
  push
             OFFFFFFFh
  call
              exit
                                    loc_8048696:
           eax, [ebp+var_10]
mov
add
           eax,
sub
           esp,
                 4
push
           eax
push
push
           [ebp+var_C]
call
add
           esp, 10h
           [ebp+var_1C], 0
[ebp+var_18], 0
mov
mov
jmp
           short loc_804870B
<u></u>
 loc_804870B:
 mov
            eax,
                  [ebp+var_18]
[ebp+var_10]
            eax,
 cmp
 j1
            short loc_80486BD
           <u></u>
           loc_80486BD:
           mov
                      eax,
                            [ebp+var_18]
           add
                      eax,
                      ecx, byte ptr [eax]
           movzx
           mov
                      edx,
                            [ebp+var_18]
                      eax, [ebp+var_14]
           mov
           add
                      eax, edx
           movzx
                      eax, byte ptr [eax]
           xor
                      eax. ecx
                      [ebp+var_1D], al
           mov
                      edx, greetingMessageax, [ebp+var_18]
           mov
           mov
           add
                      eax, edx
                      eax, byte ptr [eax]
al, [ebp+var_1D]
short loc_80486EF
           movzx
           cmp
             <mark>∭</mark> ≰ 
add
                         [ebp+var_1C],
```



There are few ways to solve this challenge:

- Reversing
- · Brute-forcing inputs

Method 1 - Reversing

read_input() (called from do_magic() just reads line from the input). The address of the string saved in var_14. The length is being saved to var 10.

A new string of the same size is being allocated (var C), and being set to zero (using memset).

var_1C and var_18 are set to zero.

Lets call the string at 0x8048858h (0x29, 0x6, 0x16, 0x4F, 0x2B, 0x35, 0x30, 0x1E, 0x51, 0x1B, 0x5B, 0x14, 0x4B, 0x8, 0x5D, 0x2B, 0x52, 0x17, 0x1, 0x57, 0x16, 0x11, 0x5C, 0x7, 0x5D, 0x0) xor_string.

Iterating over the input (with var_18 as the index):

- Set var 1D := xor string[var 18] ^ var 14[var 18].
- Compare var_1D to greetingMessage[var_18], if equal, increase var_1C (greetingMessage = You have now entered the Duck Web, and you're in for a honkin' goot time. ...)
- if var 1C equals 0x19, we won.

If out input equals to the xor of the greeingMessage with the xor_string, we win.

This simple script can generate the input for us:

```
#!/usr/bin/env python

def xor_s(s1, s2):
    s = ''

    for a,b in zip(s1, s2):
        s += chr(ord(a) ^ ord(b))

    return s

xor_string = "\x29\x06\x16\x4f\x2b\x35\x30\x1e\x51\x1b\x5b\x14\x4b\x08\x5d\x2b\x52\x17\x01\x57\x16\x11\x5c\
greetingMessage = "You have now entered the Duck Web" # This is enough for our purpose

print xor_s(xor_string, greetingMessage)
```

Output: picoCTF{qu4ckm3_7ed36e4b}

Method 2 - Brute-Forcing (like a retard)

We can see that each successful comparison increased var_1C by one.

This is done in the instruction add [ebp+var_1c], 1 which is at address 0x80486eb in the binary.

We can set a breakpoint there, brute-force one byte, and count hits (each time increase the amount of deired hits).

Using this ugly code:

```
#!/usr/bin/env python
import gdb
import string
class MyBreakpoint(gdb.Breakpoint):
    def stop (self):
       global count
       count += 1
        return False
gdb.execute('file ./main')
# Suppress output
gdb.execute('set logging file /dev/null')
gdb.execute('set logging redirect on')
gdb.execute('set logging off')
gdb.execute('set print inferior-events off')
bp = MyBreakpoint("*0x80486eB")
count = 0
flag = 'pico'
while flag[-1] != '}':
        for c in string.ascii_lowercase + string.ascii_uppercase + string.digits + '!@#$%^&*(){}_':
                gdb.execute('run < <(echo "{}") > /dev/null'.format(flag + c), to_string=True)
                if count > len(flag):
                        flag = flag + c
                        print ('Partial flag: {}'.format(flag))
                        break
gdb.execute('quit')
print ('Flag: {}'.format(flag))
```

Note: I'm sure there is a proper way doing this with pwntools, pls tell me if you know how

Method 3 - Brute-Forcing (like a boss)

Working with angr.

Installation

```
{\tt sudo \ apt-get \ install \ python 3-dev \ libffi-dev \ build-essential \ virtual envwrapper \ pip \ install \ angr}
```

Solution

TODO

Flag: picoCTF{qu4ckm3 7ed36e4b}