Protostar: stack0 write up

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

As we can see, the main function will first create **buffer** variable which contains 64 characters and initialize **modified** integer with value 0.

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

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

  modified = 0;
  gets(buffer);

  if(modified != 0) {
      printf("you have changed the 'modified' variable\n");
  } else {
      printf("Try again?\n");
  }
}
```

After that, the program will get user's input and put into **buffer** variable with **gets** function. We have to change the value of **modified** variable in order to successfully exploit the program, but since its initialization with the value 0, the program won't have anything to do with the **modified** variable again. Hmm... what could be the vulnerability here...?

Ah yes, **gets** function is the vulnerability, let's take a look at this in the **gets** function's manual:

```
Never use gets().  Because it is impossible to tell without know-
ing the data in advance how many characters gets() will read, and
because gets() will continue to store characters past the end  of
the  buffer,  it is extremely dangerous to use.  It has been used
to break computer security.  Use fgets() instead.
```

That's it, we will smash the stack.

Let's check the disassembly of the program first:

```
Dump of assembler code for function main:
0x080483f4 <main+0>:    push   ebp
0x080483f5 <main+1>:    mov    ebp,esp
0x080483f7 <main+3>:    and    esp,0xfffffff0
0x080483fa <main+6>:    sub    esp,0x60
0x080483fd <main+9>:    mov    DWORD PTR [esp+0x5c],0x0
0x08048405 <main+17>:   lea    eax,[esp+0x1c]
0x08048409 <main+21>:   mov    DWORD PTR [esp],eax
0x0804840c <main+24>:   call   0x804830c <gets@plt>
0x08048411 <main+29>:   mov    eax,DWORD PTR [esp+0x5c]
0x08048415 <main+33>:   test   eax,eax
0x08048417 <main+35>:   je     0x8048427 <main+51>
0x08048419 <main+37>:   mov    DWORD PTR [esp],0x8048500
0x08048420 <main+44>:   call   0x804832c <puts@plt>
0x08048425 <main+49>:   jmp    0x8048433 <main+63>
0x08048427 <main+51>:   mov    DWORD PTR [esp],0x8048529
0x0804842e <main+58>:   call   0x804832c <puts@plt>
0x08048433 <main+63>:   leave
0x08048434 <main+64>:   ret
```

After look at the disassembly:

+ The **gets** function will take user's input and put into **buffer** variable at **[esp+0x1c]**

+ But we need to change the value of **modified** variable at **[esp+0x5c]**

We use command *print ($esp+0x5c)-($esp+0x1c)* to calculate distance between **[esp+0x1c]** and **[esp+0x5c]**.

```
(gdb) print ($esp+0x5c)-($esp+0x1c)
$1 = 64
```

And the output of the command is 64. Voilà! So we just basically need to provide more than 64 characters to change the value of **modified** variable. Afterthat, enjoy the reward of your hard work.