1. 冷补丁：overflow

1.1 实验要求

程序 overflow 实现了一个非常简单的用户交互：输入学号，若输入的学号为 10 个字符，则在屏幕上打印一段感谢和表扬的话。程序共包含一个逻辑缺陷和一个栈溢出漏洞，要求在没有源代码的情况下对其进行修补，以满足：

- 逻辑漏洞修补后，程序仅打印与自己性别相对应的话；
- 栈溢出漏洞修补后，无论输入多长的字符串均不会导致程序崩溃。

1.2 实验过程

1.2.1 代码分析

- 首先使用 IDA 打开 overflow 程序，其 main 函数的反汇编结果如下：

- 将`main`函数反编译，结果如下：

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   char s[8]; // [rsp+5h] [rbp-Bh]
4   __int16 v5; // [rsp+Dh] [rbp-3h]
5   char v6; // [rsp+Fh] [rbp-1h]
6
7   *(_QWORD *)s = 0LL;
8   v5 = 0;
9   v6 = 0;
10  puts("Please input your student number:");
11  gets(s, argv);
12  if ( strlen(s) == 10 )
13  {
14    puts("Thank you! You are a good boy.");
15    puts("Thank you! You are a good girl.");
16  }
17  return 0;
18 }
```

- 结合以上反汇编和反编译结果，可以更加清晰地得到其执行逻辑：首先输出提示字符串，然后使用 `gets` 函数将用户的输入读取到字符数组 `s` 中，若读入字符串的长度为 `10`，则输出两条信息。与该程序的目标逻辑做对比，可知该程序存在的逻辑漏洞为输入学号后两条信息均被输出，即存在跳转的逻辑错误；存在的栈溢出漏洞为在 `gets` 函数读取用户输入时，没有对用户输入做限制，用户输入的字符串过长将导致缓冲区溢出，情况严重时会导致程序崩溃。

1.2.2 漏洞测试与分析

- 首先正常输入自己的学号，输出如下，错误地将两条提示信息全部输出，因此程序存在逻辑漏洞：

```
xubiang@kali:~/Desktop/NADP3/EXP7/WORK1
File  Actions  Edit  View  Help
[8:35:42] xubiang:WORK1 $ ./overflow.bak
Please input your student number:
U201911803
Thank you! You are a good boy.
Thank you! You are a good girl.
```

- 然后输入一个 `12` 个字符的字符串，结果如下，此时虽然栈缓冲区发生了溢出，但没有造成严重的后果，程序正常结束：

```
[8:35:47] xubiang:WORK1 $ ./overflow.bak
Please input your student number:
U20191180322
```

- 最后输入一个较长（大于等于 `19` 个字符）的字符串，结果如下，可见栈缓冲区溢出造成了段错误（或总线错误），导致了程序崩溃的严重后果，若精心构造输入的字符串，还存在其他利用栈缓冲区溢出漏洞进行执行流劫持的攻击手段，因此程序存在栈缓冲区溢出漏洞：

```
[8:35:54] xubiang:WORK1 $ ./overflow.bak
Please input your student number:
U2019118030000000000
[1]    286241 segmentation fault  ./overflow.bak
```

- 使用 `IDA` 观察 `main` 函数的栈分布可知，供字符数组 `s` 使用的栈空间为 `11` 个字节，即恰好能满足 `10` 个字符的存储需求；接下来的 `8` 个字节空间用于保存寄存器的值，若该部分被覆盖，在这个简单程序中不会造成严重的后果；再接下来的 `8` 个字节用于存放返回地址，若该部分被覆盖且覆盖后的地址不合法，则会导致段错误等问题导致程序崩溃，若输入的字符串被精心构造，则可能导致执行流被劫持，造成更严重的后果。

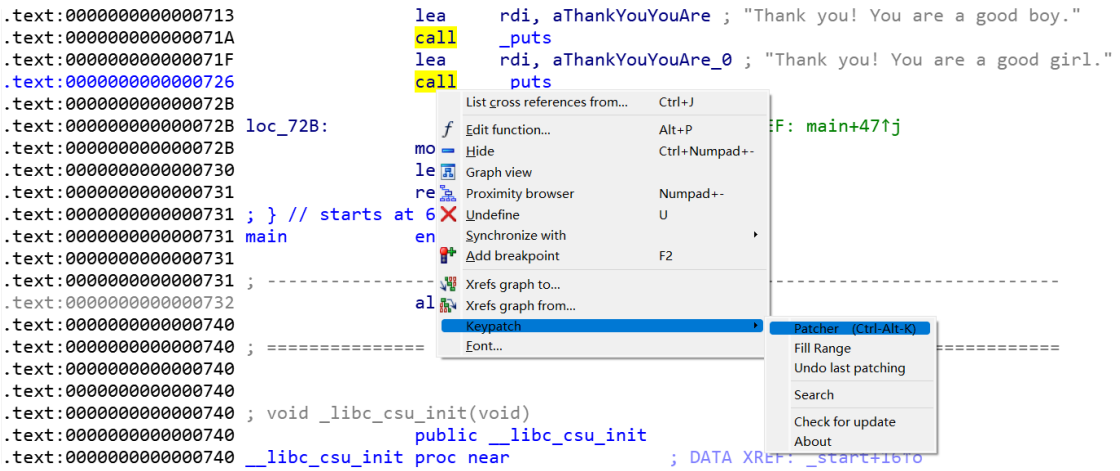
```
-0000000000000010 ; D/A/*   : change type (data/ascii/array)
-0000000000000010 ; N       : rename
-0000000000000010 ; U       : undefine
-0000000000000010 ; Use data definition commands to create local variables and function arguments.
-0000000000000010 ; Two special fields " r" and " s" represent return address and saved registers.
-0000000000000010 ; Frame size: 10; Saved regs: 8; Purge: 0
-0000000000000010 ;
-0000000000000010
-0000000000000010                 db ? ; undefined
-000000000000000F                 db ? ; undefined
-000000000000000E                 db ? ; undefined
-000000000000000D                 db ? ; undefined
-000000000000000C                 db ? ; undefined
-000000000000000B s               db 8 dup(?)
-0000000000000003 var_3           dw ?
-0000000000000001 var_1           db ?
+0000000000000000  s              db 8 dup(?)
+0000000000000008  r              db 8 dup(?)
+0000000000000010
+0000000000000010 ; end of stack variables
```

1.2.3 漏洞修补

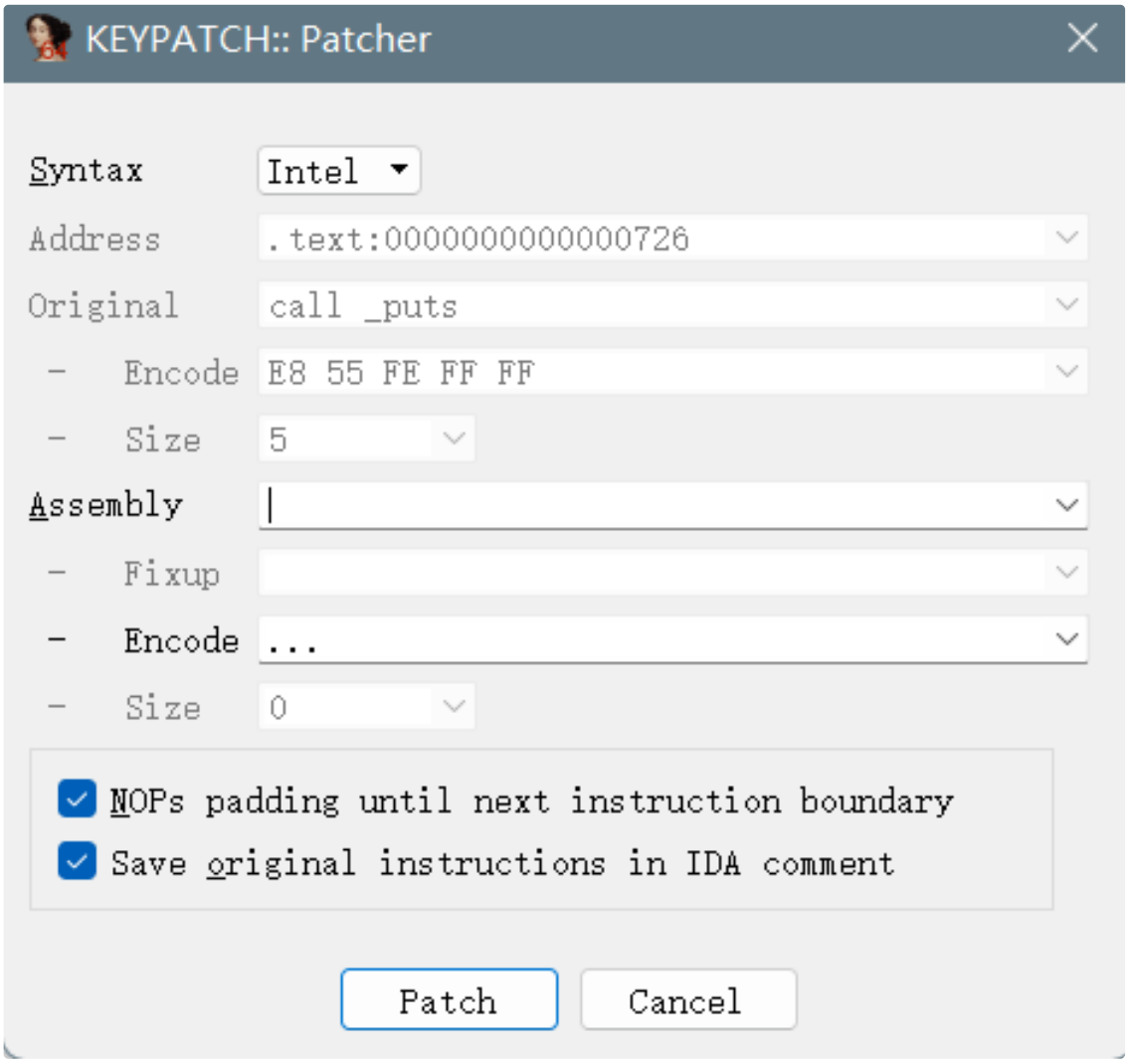
1.2.3.1 逻辑漏洞

为了仅保留两条输出语句中的第一句，可以将不需要的语句输出所对应的`lea rdi, aThankYouYouAre_0`和`call _puts`指令修改为`jmp`指令，跳转到这两条指令之后的其他指令，也可以将`call _puts`函数调用语句修改为空指令`nop`，此处采用第二中方法，具体过程如下：

- 在要修改的指令`call _puts`上右键，选择`Keypatch`中的`Patcher`功能：



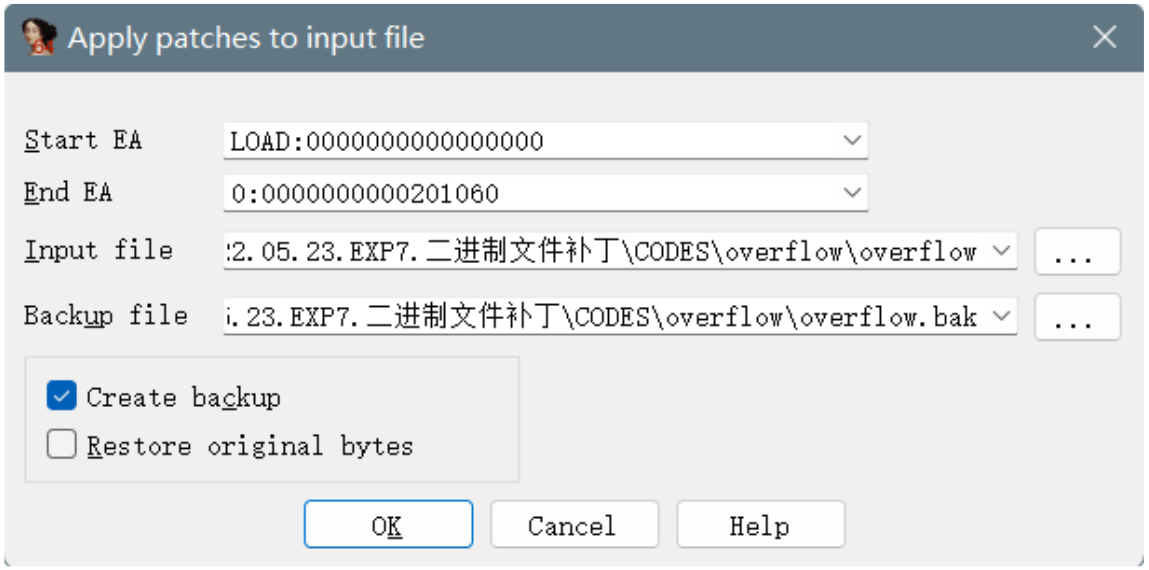
- 由于目标指令为`nop`，因此只要将`Assembly`选项留空并选择`NOPs padding until next instruction boundary`后点击`patch`即可：



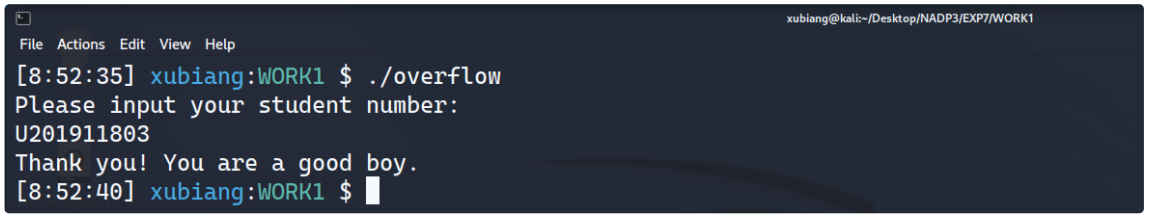
- 修改后的程序如下：

```
.text:0000000000000711                jnz     short loc_72B
.text:0000000000000713                lea     rdi, aThankYouYouAre ; "Thank you! You are a good boy."
.text:000000000000071A                call    _puts
.text:000000000000071F                lea     rdi, aThankYouYouAre_0 ; "Thank you! You are a good girl."
.text:0000000000000726                nop                     ; Keypatch modified this from:
.text:0000000000000726                                        ;   call _puts
.text:0000000000000726                                        ; Keypatch padded NOP to next boundary: 5 bytes
.text:0000000000000727                nop
.text:0000000000000728                nop
.text:0000000000000729                nop
.text:000000000000072A                nop
.text:000000000000072B
.text:000000000000072B loc_72B:                               ; CODE XREF: main+47↑j
.text:000000000000072B                mov     eax, 0
.text:0000000000000730                leave
.text:0000000000000731                retn
.text:0000000000000731 ; } // starts at 6CA
.text:0000000000000731 main           endp
```

- 在Edit中的Patch program中选择Apply patchs to input file，即可将修改应用到源文件overflow（并可选地生成备份文件overflow.bak）：



- 测试该修改后的overflow文件，输入正确的学号，结果如下，仅输出一条与boy有关的提示，逻辑漏洞被修复：



1.2.3.2 栈溢出漏洞

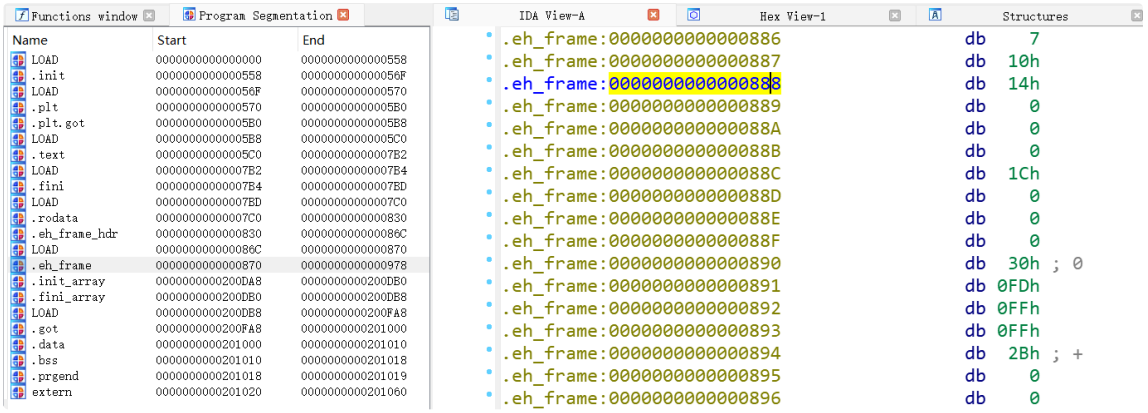
- 在修改好逻辑漏洞的程序的基础上，进一步消除栈溢出漏洞。由于gets函数本身存在不会判断输入上限的缺陷，因此选择在.eh_frame段中构造补丁代码，使用read系统调用来取代gets函数， x86_64架构下read调用的相关信息如下：

rax	System Call	rdi	rsi	rdx
0	sys_read	unsigned int fd	char* buf	size_t count

- 在IDA中查看call _get指令所在的地址为0x06FC，下一条指令的地址为0x0701：

```
.text:00000000000006EB                call    _puts
.text:00000000000006F0                lea     rax, [rbp+s]
.text:00000000000006F4                mov     rdi, rax
.text:00000000000006F7                mov     eax, 0
.text:00000000000006FC                call    _gets
.text:0000000000000701                lea     rax, [rbp+s]
.text:0000000000000705                mov     rdi, rax        ; s
.text:0000000000000708                call    _strlen
.text:000000000000070D                cmp     rax, 0Ah
.text:0000000000000711                jnz     short loc_72B
.text:0000000000000713                lea     rdi, aThankYouYouAre ; "Thank you! You are a good boy."
```

- 在 IDA 中使用 shift + F7 打开段表并查看 .eh_frame 段，选择 0x0888 处作为补丁代码的起始位置：



- 根据以上信息编写补丁代码如下，并将其使用 Keypatch 的 Patcher 或 Fill range 功能填充到 .eh_frame 段的相应位置，点击 C 将其转换为代码：

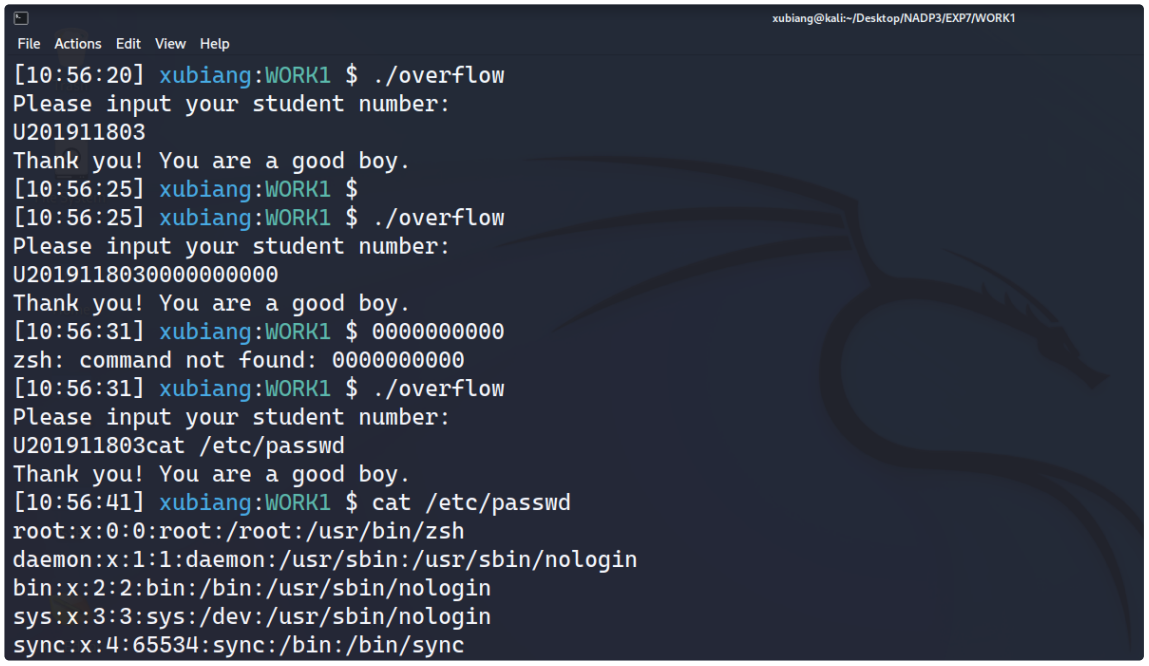
```
1  mov rax, 0           ; rax ← 0 (sys_read)
2  mov rdi, 0           ; rdi ← 0 (stdin)
3  lea rsi, [rbp-0Bh]   ; rsi ← s
4  mov rdx, 0Ah         ; rdx ← 10
5  syscall
6  jmp 0x0701           ; jmp back
```

```
.eh_frame:0000000000000888 ; ---------------------------------------------------------------------------
.eh_frame:0000000000000888                 mov     rax, 0          ; Keypatch modified this from:
.eh_frame:0000000000000888                                         ;   db 14h
.eh_frame:0000000000000888                                         ;   db 0
.eh_frame:0000000000000888                                         ;   db 0
.eh_frame:0000000000000888                                         ;   db 0
.eh_frame:0000000000000888                                         ;   db 1Ch
.eh_frame:0000000000000888                                         ;   db 0
.eh_frame:0000000000000888                                         ;   db 0
.eh_frame:000000000000088F                 mov     rdi, 0          ; Keypatch modified this from:
.eh_frame:000000000000088F                                         ;   db 0
.eh_frame:000000000000088F                                         ;   db 30h
.eh_frame:000000000000088F                                         ;   db 0FDh
.eh_frame:000000000000088F                                         ;   db 0FFh
.eh_frame:000000000000088F                                         ;   db 0FFh
.eh_frame:000000000000088F                                         ;   db 2Bh
.eh_frame:000000000000088F                                         ;   db 0
.eh_frame:0000000000000896                 lea     rsi, [rbp-0Bh]  ; Keypatch modified this from:
.eh_frame:0000000000000896                                         ;   db 0
.eh_frame:0000000000000896                                         ;   db 0
.eh_frame:0000000000000896                                         ;   db 0
.eh_frame:0000000000000896                                         ;   db 0
.eh_frame:000000000000089A                 mov     rdx, 0Ah        ; Keypatch modified this from:
.eh_frame:000000000000089A                                         ;   db 0
.eh_frame:000000000000089A                                         ;   db 0
.eh_frame:000000000000089A                                         ;   db 0
.eh_frame:000000000000089A                                         ;   db 0
.eh_frame:000000000000089A                                         ;   db 0
.eh_frame:000000000000089A                                         ;   db 0
.eh_frame:000000000000089A                                         ;   db 14h
.eh_frame:00000000000008A1                 syscall                 ; Keypatch modified this from:
.eh_frame:00000000000008A1                                         ;   db 0
.eh_frame:00000000000008A1                                         ;   db 0
.eh_frame:00000000000008A3                 jmp     loc_701         ; Keypatch modified this from:
.eh_frame:00000000000008A3                                         ;   db 0
.eh_frame:00000000000008A3                                         ;   db 0
.eh_frame:00000000000008A3                                         ;   db 0
.eh_frame:00000000000008A3                                         ;   db 0
.eh_frame:00000000000008A3                                         ;   db 0
.eh_frame:00000000000008A3 ; ---------------------------------------------------------------------------
```

- 再将源程序的`call _get`指令使用`Keypatch`修改为`jmp 0x0888`即可：

```
.text:00000000000006F0                 lea     rax, [rbp+s]
.text:00000000000006F4                 mov     rdi, rax
.text:00000000000006F7                 mov     eax, 0
.text:00000000000006FC                 jmp     loc_888         ; Keypatch modified this from:
.text:00000000000006FC                                         ;   call _gets
.text:0000000000000701 ; ---------------------------------------------------------------------------
.text:0000000000000701
.text:0000000000000701 loc_701:                                ; CODE XREF: main+1D9↓j
.text:0000000000000701                 lea     rax, [rbp+s]
.text:0000000000000705                 mov     rdi, rax        ; s
.text:0000000000000708                 call    _strlen
.text:000000000000070D                 cmp     rax, 0Ah
.text:0000000000000711                 jnz     short loc_72B
```

- 将以上修改应用到源文件并进行测试，可见无论输入多长都不会发生栈溢出，超出的字符将保留在标准输入中被后续的`shell`程序处理，栈溢出漏洞被修复（超出的字符串将被作为一条指令被`shell`执行，可能又产生了命令注入漏洞，可以通过编写更加复杂的补丁代码来解决这个问题）：


xubiang@kali:~/Desktop/NADP3/EXP7/WORK1
File Actions Edit View Help
[10:56:20] xubiang:WORK1 $ ./overflow
Please input your student number:
U201911803
Thank you! You are a good boy.
[10:56:25] xubiang:WORK1 $
[10:56:25] xubiang:WORK1 $ ./overflow
Please input your student number:
U2019118030000000000
Thank you! You are a good boy.
[10:56:31] xubiang:WORK1 $ 0000000000
zsh: command not found: 0000000000
[10:56:31] xubiang:WORK1 $ ./overflow
Please input your student number:
U201911803cat /etc/passwd
Thank you! You are a good boy.
[10:56:41] xubiang:WORK1 $ cat /etc/passwd
root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync