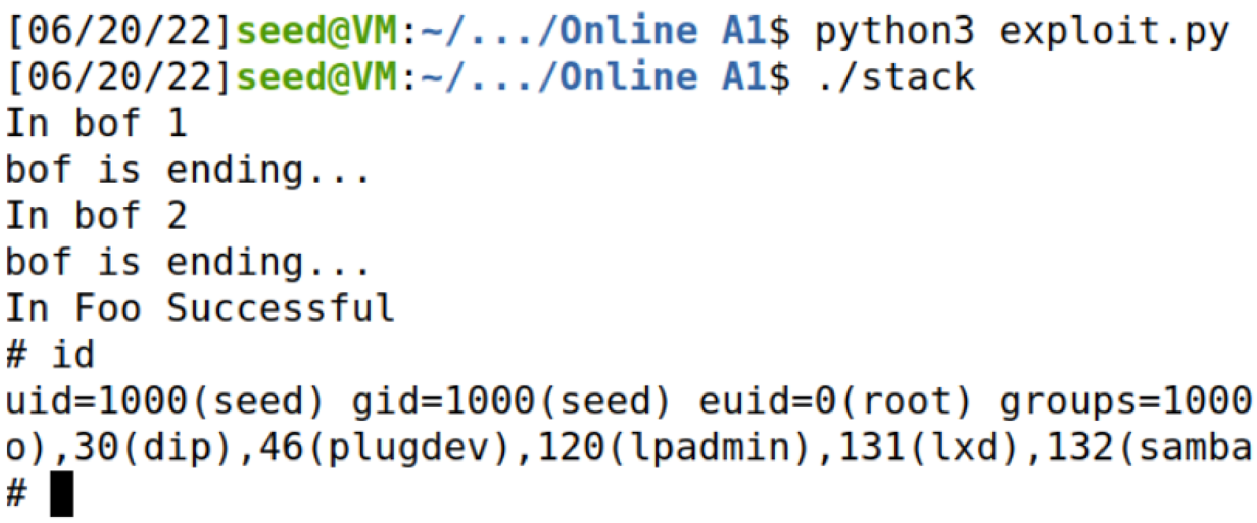
CSE 406 (July 2022)
Buffer Overflow Online (A - 1)

You are given the following vulnerable C program *A1.c*. Replace <param_1> and <param_2> in the source code with the corresponding values of Table-1.

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

#ifndef BUF_SIZE1
#define BUF_SIZE1 <param_1>
#endif

int check_a = <param_2>, check_time = 1;
char* str_1;
FILE *badfile;

char code[] =
  "\x31\xc0"             /* xorl    %eax,%eax              */
  "\x50"                 /* pushl   %eax                   */
  "\x68""//sh"           /* pushl   $0x68732f2f            */
  "\x68""/bin"           /* pushl   $0x6e69622f            */
  "\x89\xe3"             /* movl    %esp,%ebx              */
  "\x50"                 /* pushl   %eax                   */
  "\x53"                 /* pushl   %ebx                   */
  "\x89\xe1"             /* movl    %esp,%ecx              */
  "\x99"                 /* cdq                            */
  "\xb0\x0b"             /* movb    $0x0b,%al              */
  "\xcd\x80"             /* int     $0x80                  */
;

int bof(int a, char *guard_str, int time)
{
    int c; char localstr[10] = "Dest";
    c = 6;
    char buffer[BUF_SIZE1];
    strcpy(buffer,localstr);
    printf("In bof %d\n",time);
    strcpy(buffer, guard_str);
    if (a != check_a || time != check_time){
        printf("Try Again.");
        exit(1);
    }
    printf("bof is ending...\n");
    return 0;
}

int foo(int c, int a, int b){
    if(a == 768 && b == 68){
        printf("In Foo Successful\n");
        ((void(*)( ))code)( );
    }
    //printf("%d %d\n",a,b);
    return 0;
}

int main(int argc, char **argv)
{
    str_1 = (char*)malloc(800 * sizeof(char));

    badfile = fopen("badfile", "r");
    if (!badfile) {
       perror("Opening badfile");
       exit(1);
    }
    fread(str_1, sizeof(char), 800, badfile);
    bof(check_a, "Normal Test", check_time);
    check_time++;
    bof(check_a, str_1, check_time);
    fprintf(stdout, "==== Returned Properly ====\n");
    return 0;
}
```

Tasks:

1. First, compile the program with the 32 bit flag set as demonstrated in the class. Do not forget to turn off address space randomization and stack protection. Also, make sure that the stack is executable while compiling the program.
2. Prepare a payload (e.g. badfile) which will cause the program to open a shell with root's privilege and print exactly the lines shown in the figure below



```
[06/20/22]seed@VM:~/.../Online A1$ python3 exploit.py
[06/20/22]seed@VM:~/.../Online A1$ ./stack
In bof 1
bof is ending...
In bof 2
bof is ending...
In Foo Successful
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000
o),30(dip),46(plugdev),120(lpadmin),131(lxd),132(samba
# █
```

.

3. Rename your ***exploit.py*** file with ***1705XXX.py*** and submit on moodle.

Marks Distribution

Item	**Marks**
Escaping security measure of ***bof***	5
Calling ***foo***	4
Opening the shell	6
Viva	5
Total	**20**

Table 1

Student ID	param_1	param_2
1605021	364	52
1605036	461	51
1605051	381	32
1605085	386	46
1705001	432	38
1705002	438	49
1705003	447	52
1705004	499	46

Student ID	param_1	param_2
1705005	352	49
1705006	443	85
1705007	444	49
1705008	437	78
1705009	474	35
1705010	367	72
1705011	443	47
1705012	449	45
1705013	404	47
1705014	363	53
1705015	497	23
1705016	365	31
1705017	381	57
1705018	405	54
1705019	477	60
1705020	406	24
1705021	478	70
1705022	387	69
1705023	464	50
1705024	487	23
1705025	407	60
1705026	434	51
1705027	386	46
1705028	377	36
1705029	412	28
1705030	416	57
1705031	377	35
1705039	403	69