PSG College of Technology
Department of Applied Mathematics and Computational Sciences
MSc Theoretical Computer Science – VI Semester
15XT68 -Security in Computing Lab
Problem Sheet 5

Detection of a Buffer overflow attack

```
/* stack.c */

/* This program has a buffer overflow vulnerability. */
/* Our task is to exploit this vulnerability */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int bof(char *str)
{
char buffer[12];
/* The following statement has a buffer overflow problem */
strcpy(buffer, str);
return 1;
}
int main(int argc, char **argv)
{
char str[517];
FILE *badfile;
badfile = fopen("badfile", "r");
fread(str, sizeof(char), 517, badfile);
bof(str);
printf("Returned Properly\n");
return 1;
}
```

Compile the above vulnerable program and make it set-root-uid. The above program has a buffer overflow vulnerability. It first reads an input from a file called "badfile" ,and then passes this input to another buffer in the function `bof()`. The original input can have a maximum length of 517 bytes, but the buffer in `bof()` has only 12 bytes long. Because `strcpy()` does not check boundaries, buffer overflow will occur. Since this program is a set-root-uid program, if a normal user can exploit this buffer overflow vulnerability, the normal user might be able to get a root shell. It should be noted that the program gets its input from a file called "badfile". This file is under users' control. Now, our objective is to create the contents for "badfile", such that when the vulnerable program copies the contents into its buffer, a root shell can be spawned.

Write a Java program with command line arguments. Include a function exploitable() to check buffer overflow attack. Hint: Try copying the arg value to a char array of smaller size.