**NAME *: LAKSHITHA RAJ VASANADU***

**ID *: 00001115006***

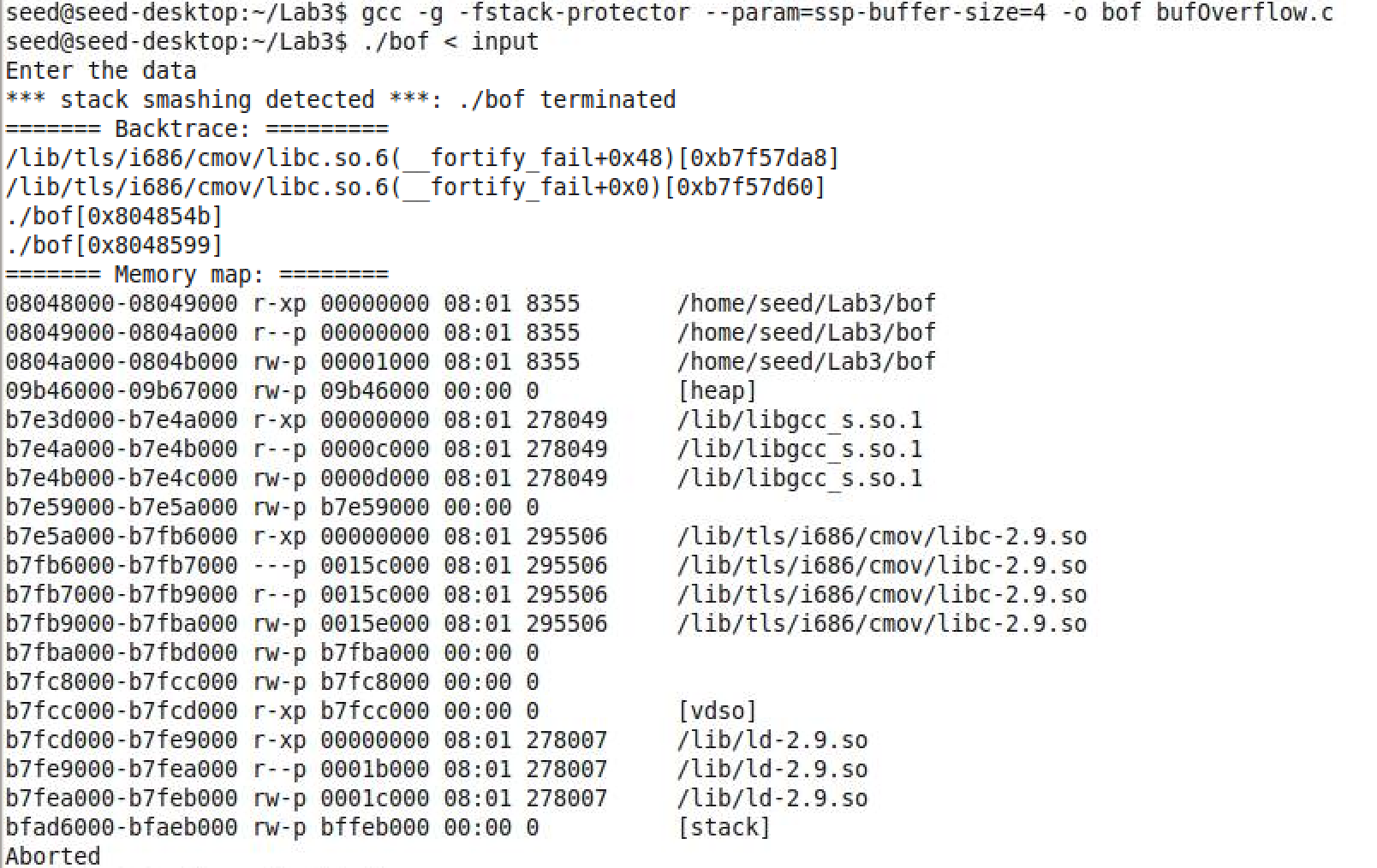
**LAB ASSIGNMENT 3 TASK 1**

**MITIGATION STRATEGIES AND EXPLORING THE STACK**

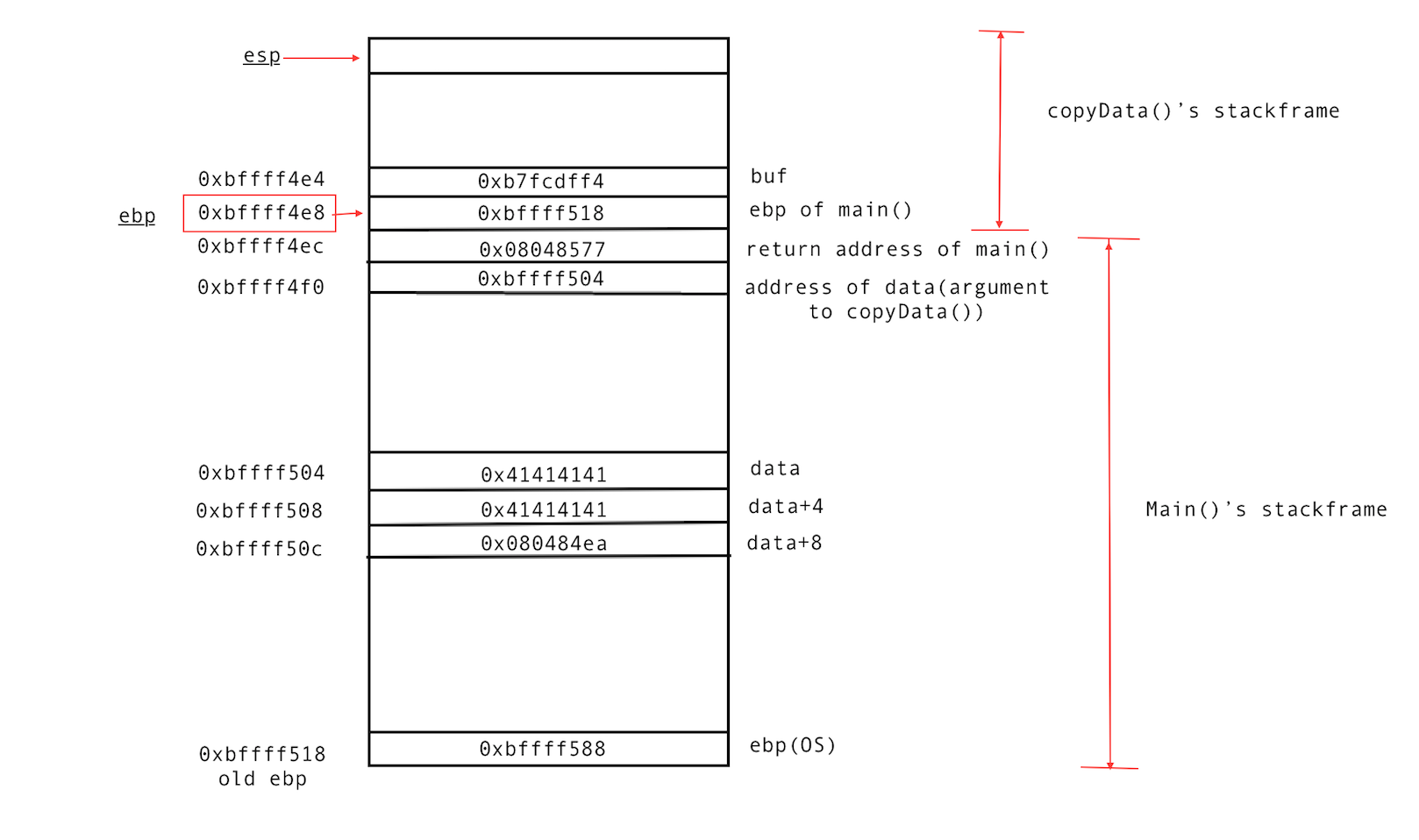
1. Both –fstack-protector and –fstack-protector-all do not work with the given program.

* **-fstack-protector** : This option emits extra code to check for buffer overflows by adding a guard variable to functions with vulnerable objects like *alloca* or buffers with size > 8 bytes.
* **-fstack-protector-all** : This does the same as above but checks for stack smashing in every function.

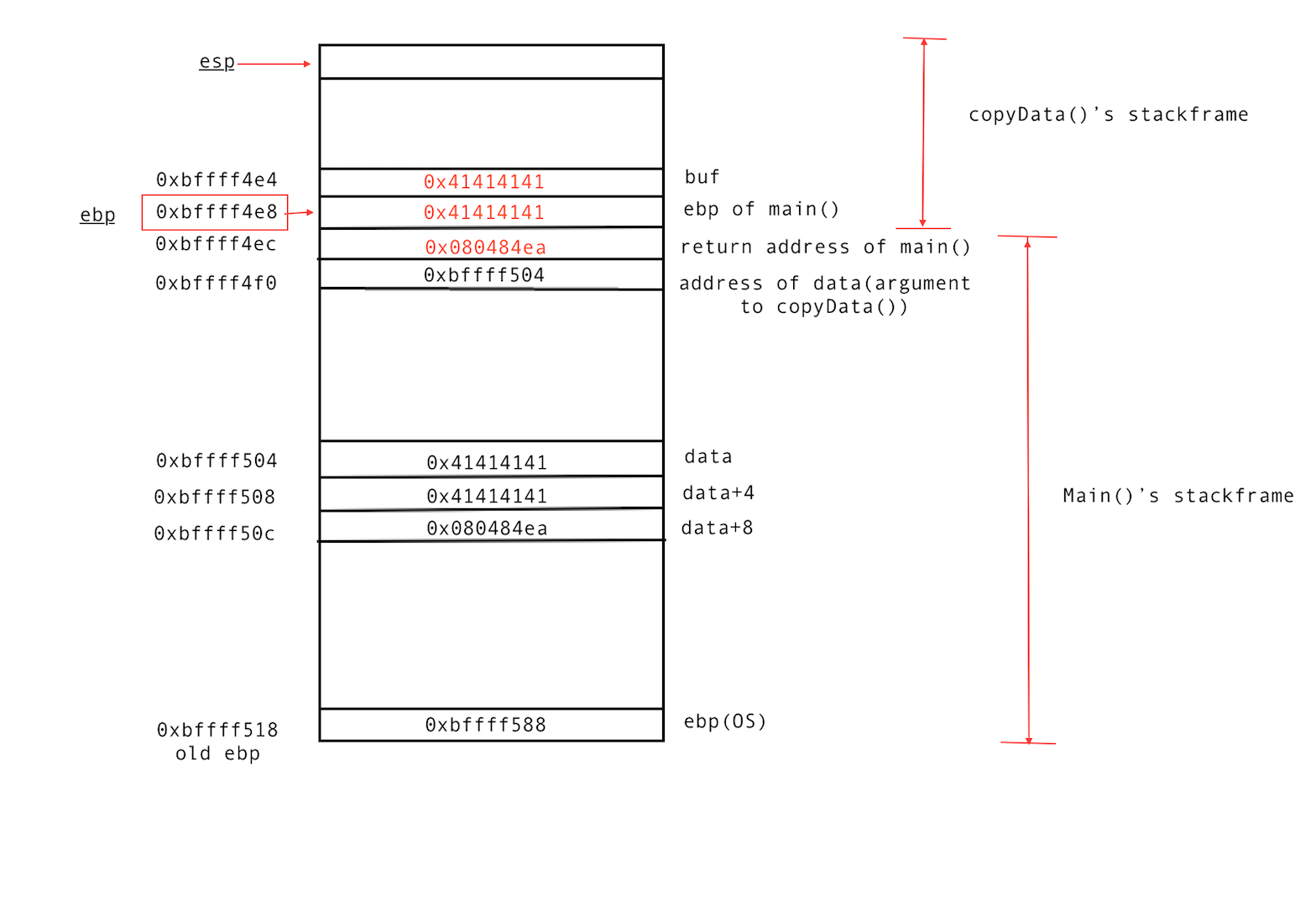
Both of these work only for buffer > 8 bytes. In the given code, buffer size is 4 (**char buf[4]**) and hence it does not work. Instead, the option **--param=ssp-buffer-size=4** can be used with gcc to explicitly specify the buffer size. This helps to protect the code from buffer overflows.



1. **Before strncpy()**



**After strncpy()**

****

Here, we can observe that return address in main() is over written by our input.

1. The given program can be re-written as follows to protect from buffer overflows. This code uses **Input Validation** as a mitigation strategy to **prevent** buffer overflow.

Here, we check whether the length of the input parameter to *CopyData* function is less than the buffer size that is statically allocated. If the condition fails, the program in aborted.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void message() {

puts(“Should not reach here”);

exit(0);

}

void copyData(char\* str) {

char buf[4];

if(strlen(str) >= sizeof(buf)) {

printf(“Aborting the program due to long input!!”);

abort();

}

strncpy(buf, str, strlen(str));

}

int main(int argc, char\* argv[]) {

char data[12];

printf(“Enter the data\n”);

scanf(“%s”, data);

copyData(data);

return 0;

}