Santa Clara University
Computer Engineering Department

COEN 225 Secure Coding in C and C++

GDB Lab

***Objectives*:** 1. To learn the Gnu Debugger (gdb) commands used to trace program execution.
2. To draw a diagram of the stack showing the addresses and data stored on it at a particular instant of execution.

Setup Procedure*:*

1. Login as root and turn off address randomization in the virtual machine.

2. Compile the program below (called prog.c) using gcc with the **–g** flag as follows:
   gcc –g –o prog prog.c

prog.c:

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

void foo(int x, char *buf);

void foo(int x, char *buf) {
   printf("%d\n", x);
   strcpy(buf, "ABCDEFG");
}


int main() {
  int x1 = 10;
  char buffer[20] = "abcdefg";
  foo(x1, buffer);
  printf("%s", buffer);
  return 0;
}
```

3. Open the executable inside gdb using the command:
   gdb prog

4. Now, follow the directions and answer the questions on the following pages.

Examining the program inside gdb:

---

1. View the source code for main():
   list main

2. View the assembly instructions for main():
   disas main

3. View the assembly instructions for foo():
   disas foo

4. Display the address of main():
   p main

Answer the following questions:

- What is the address of the instruction to call printf() in foo()?
  Answer: *disas foo*
  0x08048468

- What is the address of the function foo()?

Answer: *p foo*
0x0804854

- What is the address of the function printf()?

Answer: p printf
No symbol printf in the current context

The address of printf() is not known currently. It will be determined by the runtime linker when the function is called the first time.

Setting breakpoints and running the program inside gdb:

---

1. To set a breakpoint on line 15 of main, use:
   b 15

   (You can view the line numbers using *list main*)

```
12      int main() {
13          int x1 = 10;
14          char buffer[20] = "abcdefg";
15          foo(x1, buffer);
```

2. Run the program:
   run

The program execution will stop at line 15.

3. To view assembly instructions while stepping the program, use:
   disp/i $pc

4. Now step one instruction at a time, using
   si

5. Your program output will look like this:

```
(gdb) b 15
Breakpoint 1 at 0x80484d0: file prog.c, line 15.
(gdb) run
Starting program: /home/seed/prog

Breakpoint 1, main () at prog.c:15
15        foo(x1, buffer);
(gdb) disp/i $pc
1: x/i $pc
0x80484d0 <main+70>:    lea    -0x1c(%ebp),%eax
(gdb) si
0x080484d3      15        foo(x1, buffer);
1: x/i $pc
0x80484d3 <main+73>:    mov    %eax,0x4(%esp)
(gdb) si
0x080484d7      15        foo(x1, buffer);
1: x/i $pc
0x80484d7 <main+77>:    mov    -0x20(%ebp),%eax
(gdb) si
0x080484da      15        foo(x1, buffer);
1: x/i $pc
0x80484da <main+80>:    mov    %eax,(%esp)
```

The function foo() has been called, and so the two mov statements are setting up the arguments on the stack. Recall that the arguments are set up using the ebp register. The first argument is at address **(ebp – 0x1c)**, and the second argument is at address **(ebp – 0x20)**. Now, you will see how to examine memory to find out what is the data value at these addresses.

Examining registers and memory inside gdb:

---

1. Examine the contents of a register as follows:
   x $ebp

```
(gdb) x $ebp
0xbffff538:     0xbffff5a8
```

This means that register ebp contains 0xbffff538. The data stored in memory at address 0xbffff528 is 0bbfff5a8.

2. Examine the contents of a memory location (say 0xbffff538) as follows:
   x 0xbffff538

```
(gdb) x 0xbffff538
0xbffff538:     0xbffff5a8
```

3. To determine the contents of all registers, use :
   i r

Answer the following questions:

- What is the data stored at address ebp – 0x1c?

Answer: 0x64636261

Note that these are ascii values of d(64), c(63), b(62), a (61). These are the first four bytes of `buffer` in prog.c.

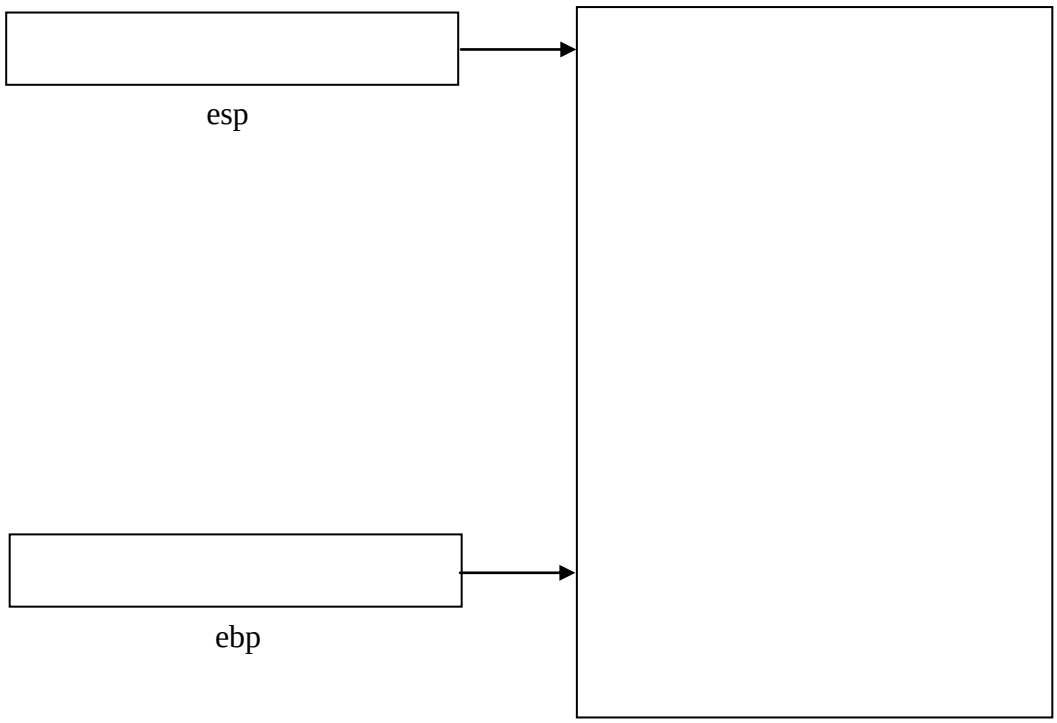
- What is the data stored at address ebp – 0x20?

Answer: 0x0000000a

This is the value of x1 =10.

- What addresses are stored in ebp and esp?
  ebp contains 0xbffff538 and esp contains 0xbffff500

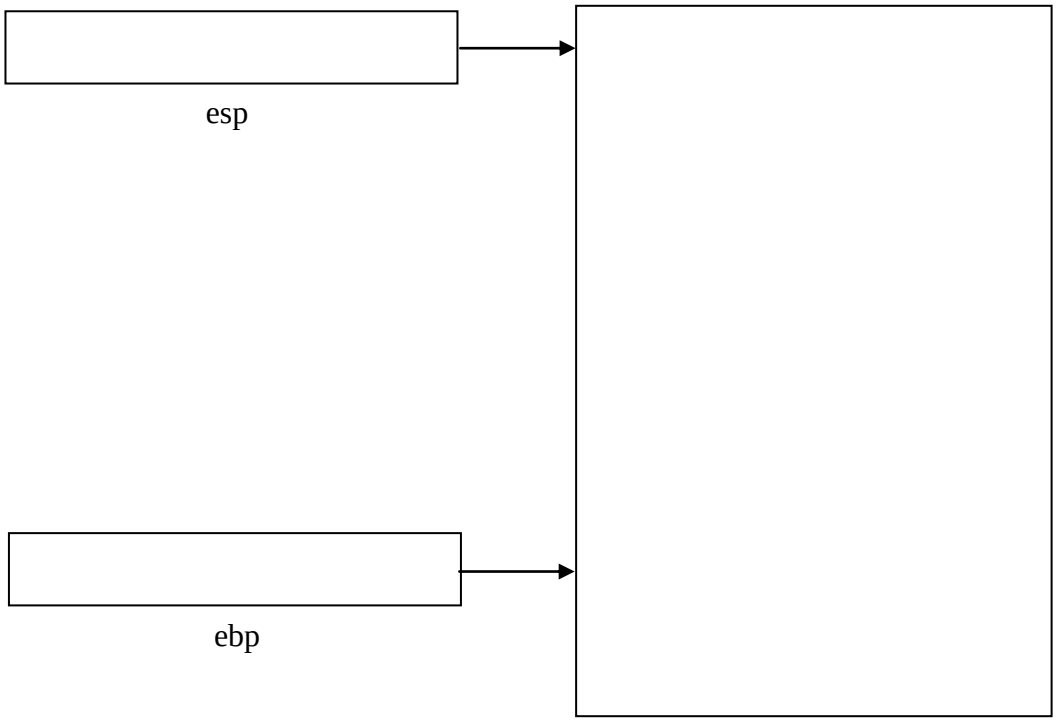
- Draw a diagram of the stack at this point. Show addresses in esp and ebp and the data values these point to. Also, show where buffer and x1 are stored.



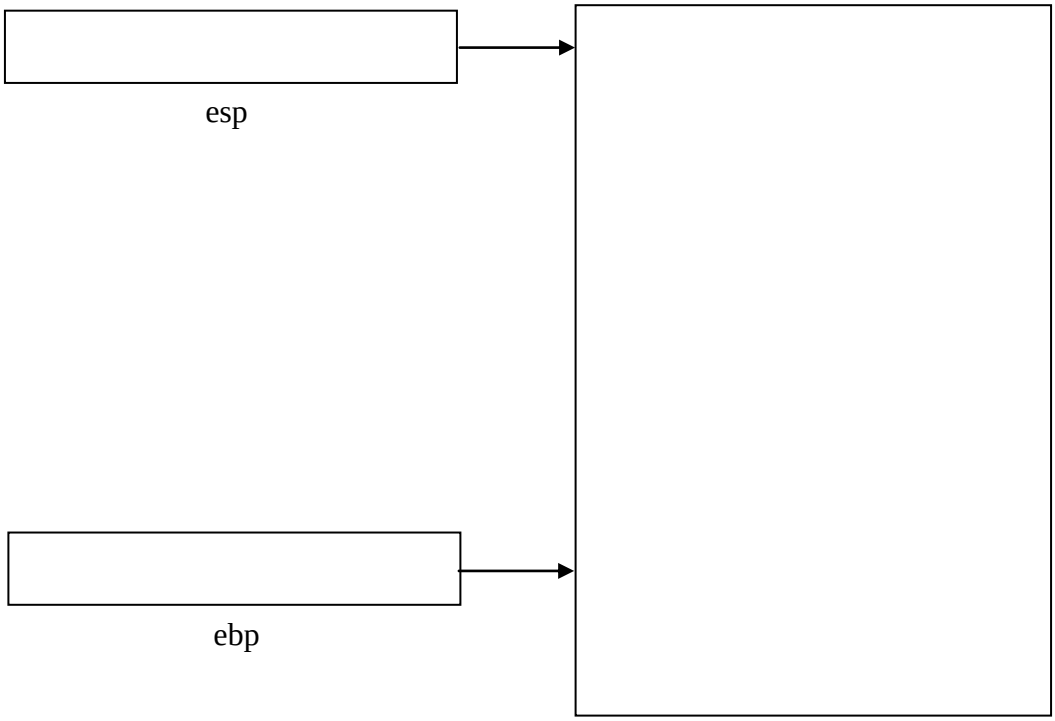
Tracing function execution inside gdb:

4. Continue stepping through the instructions using the *si* command (or simply hitting Enter). After each step, redraw the stack diagram to show what has changed in that step. Explain your observation.

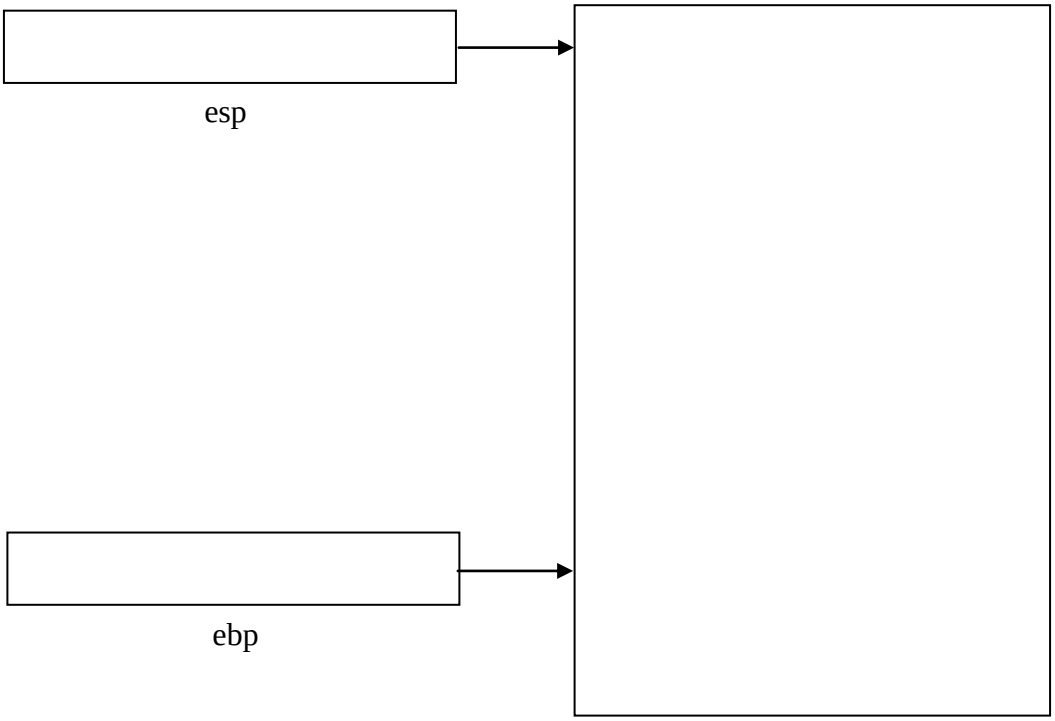
```
1: x/i $pc
0x80484dd <main+83>:    call   0x8048454 <foo>
```



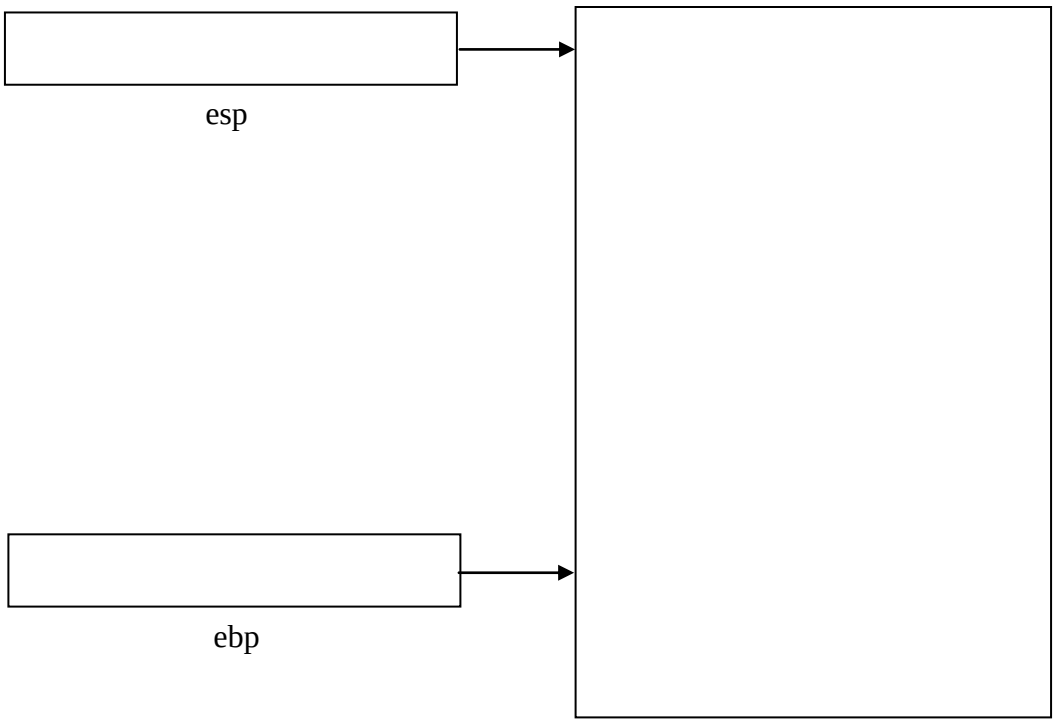
```
1: x/i $pc
0x8048454 <foo>:        push   %ebp
```



```
1: x/i $pc
0x8048455 <foo+1>:      mov    %esp,%ebp
```



```
1: x/i $pc
0x8048457 <foo+3>:      sub    $0x18,%esp
```



Now the printf function is being called. Since this contains a large number of instructions, you should use n to complete executing this with one command.

```
(gdb) si
7           printf("%d\n", x);
1: x/i $pc
0x804845a <foo+6>:      mov    0x8(%ebp),%eax
(gdb) n
10
8           strcpy(buf, "ABCDEFG");
1: x/i $pc
0x804846d <foo+25>:     movl   $0x8,0x8(%esp)
```

- Since printf has executed once, it has been linked from the library into the program. What is this address of this printf() function?

- What happens after strcpy executes. Show the changes in main's stack frame. Show the contents of the esp and ebp registers.