Harvey Mudd College Name: _____
CS 105, Computer Systems, Fall 2012
QUIZ 4 —— CLOSED BOOK, CLOSED NOTEs

This Quiz concerns the following C code:

```
/* copy string x to buf */
void foo(char *x) {
  int buf[1];
  strcpy((char *)buf, x);
}

void callfoo() {
  foo("abcdefghi");
}
```

Here is the corresponding machine code on a Linux/x86 machine:

```
080484f4 <foo>:
080484f4: 55              pushl  %ebp
080484f5: 89 e5           movl   %esp,%ebp
080484f7: 83 ec 18        subl   $0x18,%esp
080484fa: 8b 45 08        movl   0x8(%ebp),%eax
080484fd: 83 c4 f8        addl   $0xfffffff8,%esp
08048500: 50              pushl  %eax
08048501: 8d 45 fc        leal   0xfffffffc(%ebp),%eax
08048504: 50              pushl  %eax
08048505: e8 ba fe ff ff  call   80483c4 <strcpy>
0804850a: 89 ec           movl   %ebp,%esp
0804850c: 5d              popl   %ebp
0804850d: c3              ret

08048510 <callfoo>:
08048510: 55              pushl  %ebp
08048511: 89 e5           movl   %esp,%ebp
08048513: 83 ec 08        subl   $0x8,%esp
08048516: 83 c4 f4        addl   $0xfffffff4,%esp
08048519: 68 9c 85 04 08  pushl  $0x804859c   { push string address}
0804851e: e8 d1 ff ff ff  call   80484f4 <foo>
08048523: 89 ec           movl   %ebp,%esp
08048525: 5d              popl   %ebp
08048526: c3              ret
```

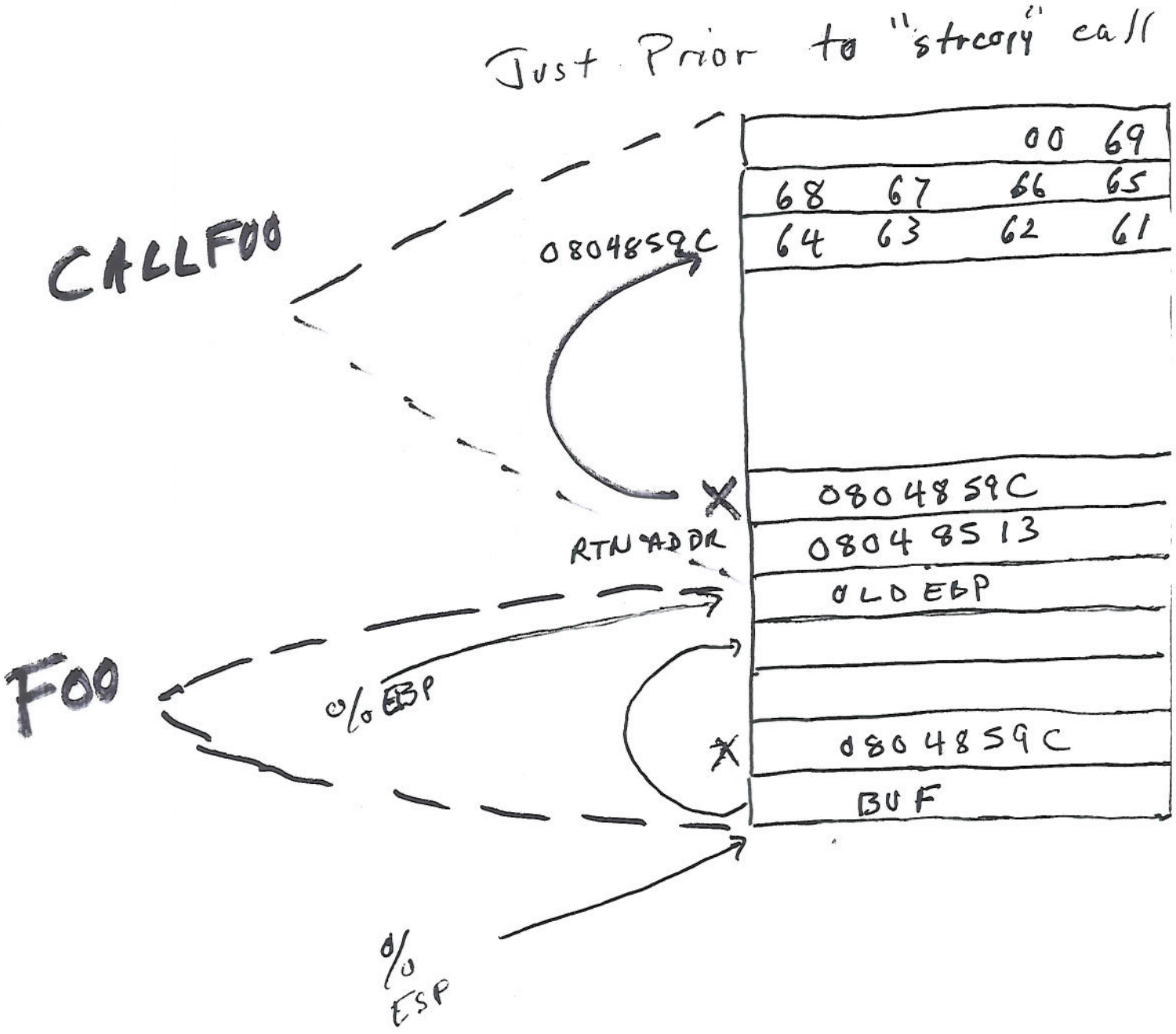
This is an example of buffer overflow. There are a number of 'points'
to notice:

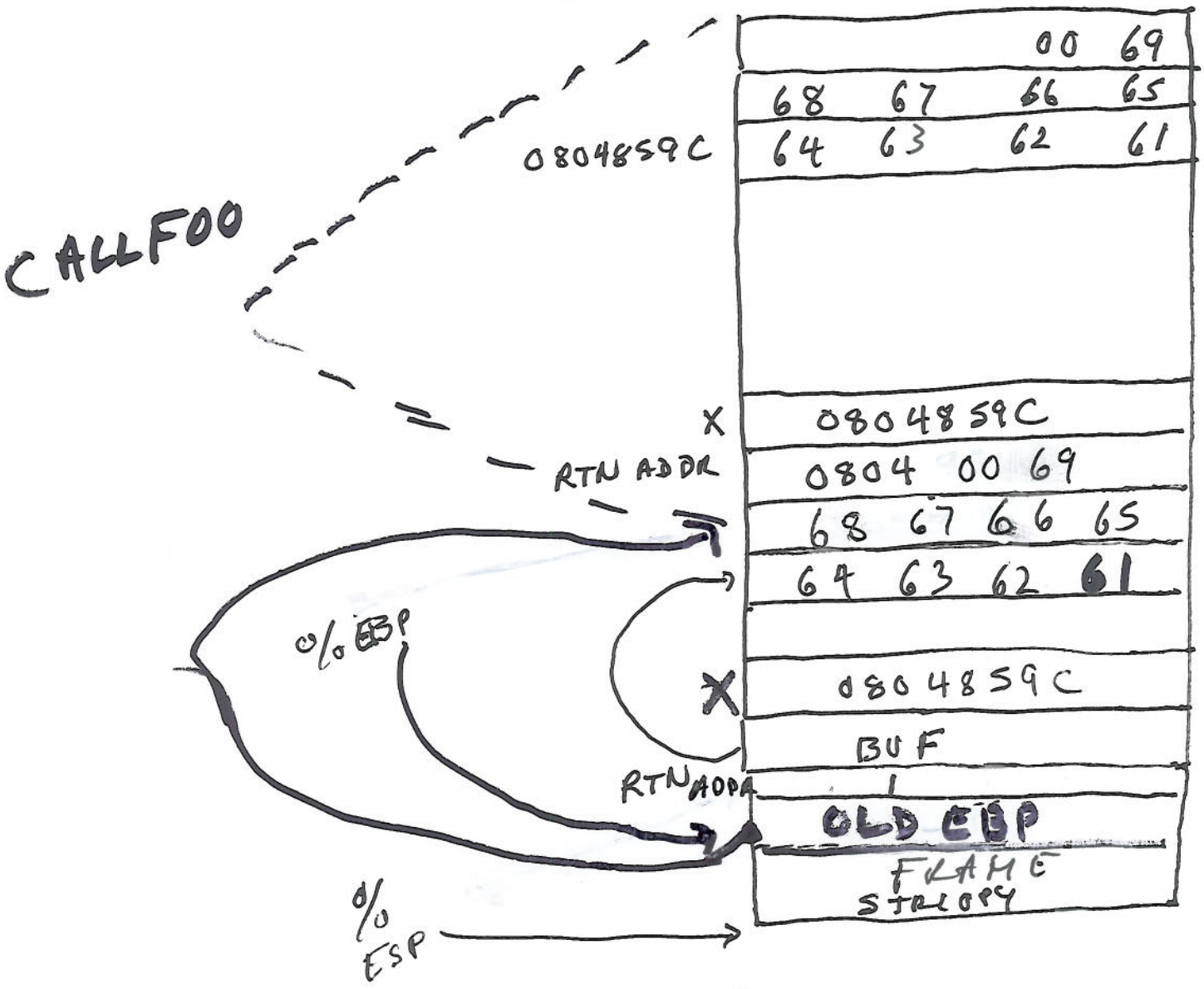
1. For whatever reason the compiler adds stack space to the Foo frame:
    Instuction: 4fd

2. Foo pushes, the pointer X onto the stack for the call to strcpy:
    4fa and 500

3. Foo creates and pushes the address for Buf onto the stack:
    501 and 504.

4. strcpy is going to write into Foo's stack frame given the starting
    address specified by Buf. Keys to note:
      1. the first character goes into the Least Significant Byte of Buf
      2. we continue putting characters one after the other in an 'upwards'
          direction.
      3. since Buf is only 4 bytes, we will overwrite the 'OLD EBP' completely
          and overwrite the 2 Least Significant Bytes of the Return Address.


Just Prior to "strcpy" call
CALLFOO
08048598
FOO
%EBP
%ESP
RTN ADDR
OLD EBP
BUF
00 69
68 67 66 65
64 63 62 61
08048598
08048513
08048598



Just Before Return from str copy

Once Return from strcopy

%ebp = 68 67 66 65

%eip = 08 04 00 69

This problem tests your understanding of the stack discipline and byte ordering. Here are some notes to help you work the problem:

- **if you hope to solve this problem, then you need to draw yourself a picture of the stack at the point that strcpy is called showing foo's frame**
- `strcpy(char *dst, char *src)` copies the string at address `src` (including the terminating '\0' character) to address `dst`. It does **not** check the size of the destination buffer.
- Recall that Linux/x86 machines are Little Endian.
- You will need to know the hex values of the following characters:

Character	Hex value	Character	Hex value
'a'	0x61	'f'	0x66
'b'	0x62	'g'	0x67
'c'	0x63	'h'	0x68
'd'	0x64	'i'	0x69
'e'	0x65	'\0'	0x00

Now consider what happens on a Linux/x86 machine when `callfoo` calls `foo` with the input string "`abcdefghi`".

- List the contents of the following memory locations immediately after `strcpy` returns to `foo`. Each answer should be an unsigned 4-byte integer expressed as 8 hex digits.

`buf[0]` = 0x 64636261

`buf[1]` = 0x 68676665

`buf[2]` = 0x 08040069

- Immediately **before** the `ret` instruction at address `0x0804850d` executes, what is the value of the frame pointer register `%ebp`?

`%ebp` = 0x 68676665

- Immediately **after** the `ret` instruction at address `0x0804850d` executes, what is the value of the program counter register `%eip`?

`%eip` = 0x 08040069