### Practice Problem 8.1

What feature of the C language makes a buffer overflow attack possible?

Solution:

C compilers do not check for reading or writing beyond the bounds of an array. This is a feature because C programs typically execute faster than programs written in Java, Python, or C++, because they spend less time checking boundaries. It is also a vulnerability which can lead to buffer overflow attacks if the programmer is not careful.

### Practice Problem 8.2

Describe the mechanism by which a segmentation fault occurs.

Solution:

Computers can execute many different programs simultaneously. One of the jobs of the operating system is to assign blocks, or segments of memory to each running program. When a program attempts to access a memory location outside of its assigned segment, the operating system will close (kill) the program and report a segmentation fault.

A segfault can also occur when a program attempts to write to a read-only portion of its own memory segment, for example the text segment is marked as read-only.

For example, if a program declares a pointer without initializing it, and then reads or writes data using that "wild" pointer, it may produce a segfault.

Another example: if a function’s return address is overwritten by a buffer overflow, this will cause a segfault if the program attempts to return to an address outside its segment.
**Practice Problem 8.3**

For the `pawn` function below, is it possible to overwrite the value of the variable `cost` by entering a string beyond the bounds of the array `item` on the stack during the `scanf` function call? Explain.

```c
void pawn()
{
    char item[12];
    int cost = 100;
    printf( "What have you come to sell?" );
    scanf( "%s", item );
}
int main()
{
    pawn();
}
```

Solution:

No. Because `item[]` is declared prior to `cost`, `item` will be on the bottom of the stack, and `cost` will be higher on the stack. An overflow on `item[]` will overwrite higher address, but not lower addresses.

**Practice Problem 8.4**

When the `echo_string()` function is called in `main()` from the following code sample, the stack pictured below is created. Note: `main()` has no variables declared, and the function takes no arguments.

```c
#include<stdio.h>
void echo_string()
{
    int count;
    char entered_string[10];
    printf("Enter a string: ");
    scanf("%s", entered_string);
    for(count=0; count<10; count=count+1)
    {
        printf("%s\n",entered_string);
    }
}
int main()
{
    echo_string();
}
```

Assuming there is no padding (extra spaces) when the frame is created. How many characters can be entered before the return address is changed? Completely overwritten?

Solution:

When 18 characters are entered, 19 bytes are written, where the 19th byte is the NULL 0x00. The first 18 will fully occupy the 10 bytes allocated for the array `entered_string[10]`, plus all 4 bytes of `count`, and all 4 bytes of prior ebp. The 19th byte will overwrite the first byte (least significant byte) of the return address with 0x00.
When 21 characters are entered, 22 bytes will be written, which will completely overwrite the following:
entered_string  10 bytes
count          4 bytes
prior ebp      4 bytes
return address 4 bytes
------------------------
Total          22 bytes