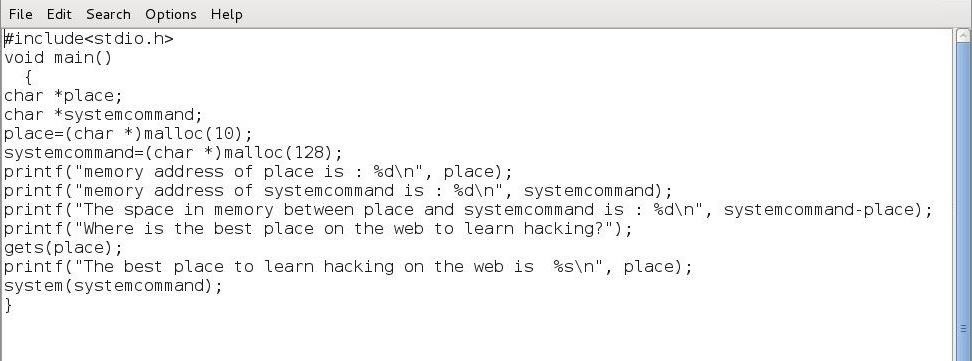
Write the Code

I have developed a small bit of code that will enable us to overflow a memory buffer and run our own commands on the system. It is not meant to be used as an exploit, but rather to simply demonstrate the principle of buffer overflows that we will be building into our zero-day exploit.

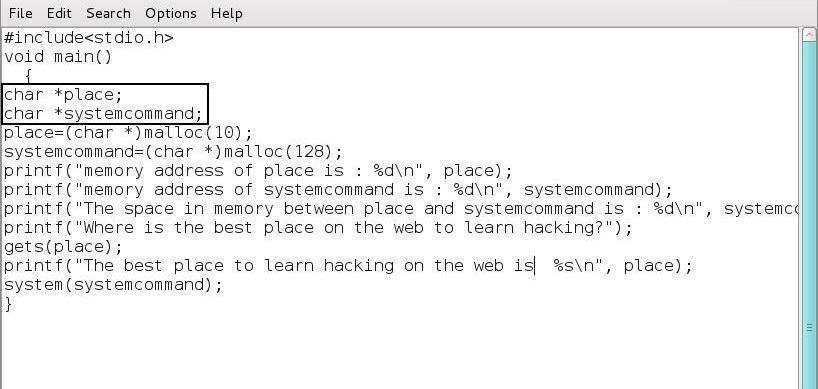
Now, enter the following code as shown below.

[](https://img.wonderhowto.com/img/original/77/35/63556333295063/0/635563332950637735.jpg)

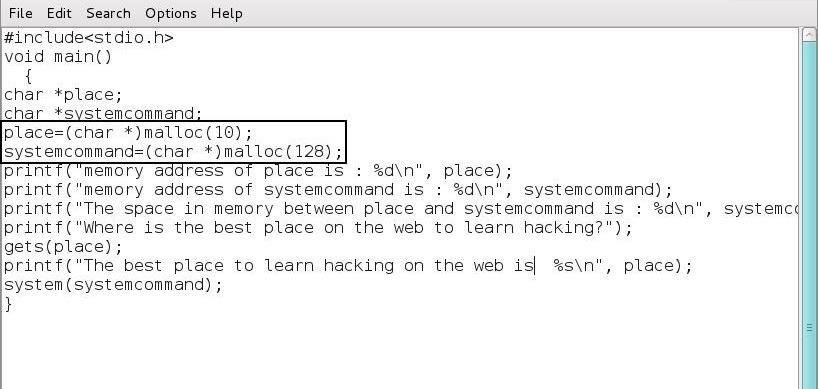
In the screenshot below, I have highlighted our two variables we will be using in this code;

***char \*place***  
***char \*systemcommand***

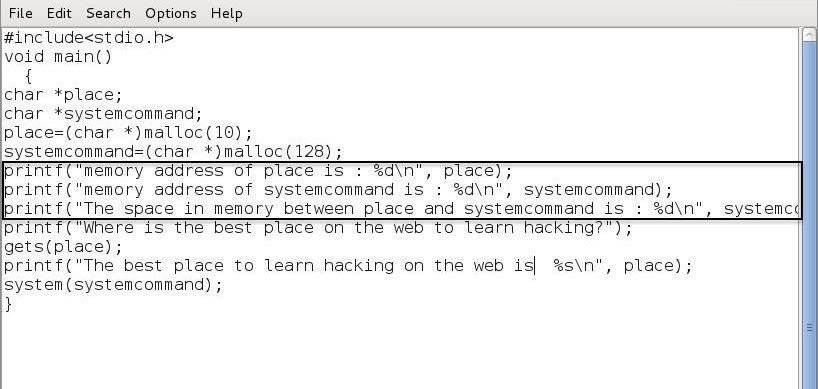
We have declared them below both "char" or character type variables.

[](https://img.wonderhowto.com/img/original/39/85/63556312352954/0/635563123529543985.jpg)

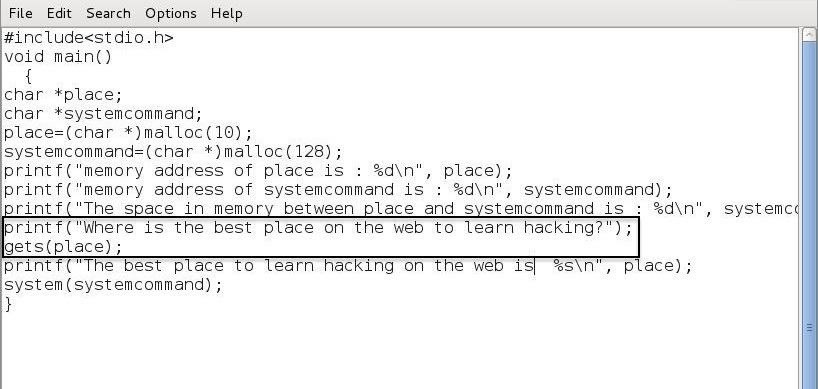
Next, we have allocated memory for each variable using the malloc (memory allocation) command.

[](https://img.wonderhowto.com/img/original/42/46/63556312366579/0/635563123665794246.jpg)

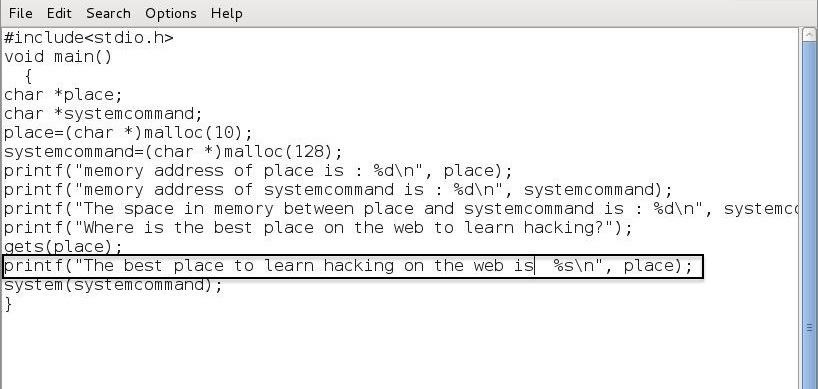
After the memory allocation, we have two "printf" statements that print the memory locations of the two variables. The third printf below then calculates the number of bytes between the two memory locations of our variables.

[](https://img.wonderhowto.com/img/original/01/72/63556313867970/0/635563138679700172.jpg)

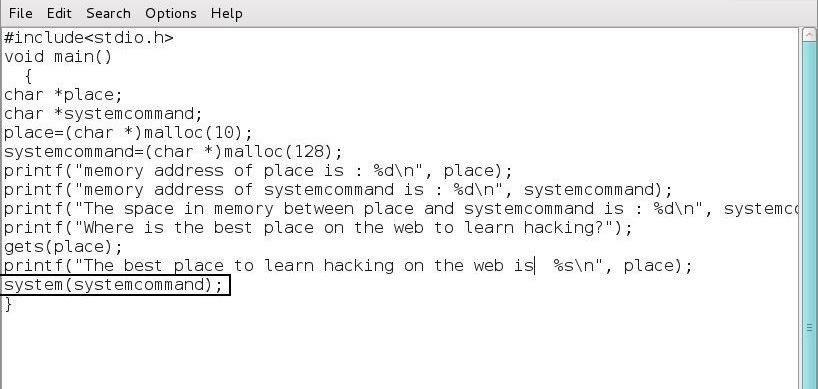
The fourth printf then asks the user "Where is the best place to learn hacking on the web?" followed by the "gets" function that puts the users response into the variable "place."

[](https://img.wonderhowto.com/img/original/13/17/63556313927673/0/635563139276731317.jpg)

This is then followed by another printf function that prints the statement "The best place to learn hacking on the web is" followed by the user's response. Of course, the user will respond with "Null Byte"... what else?

[](https://img.wonderhowto.com/img/original/70/86/63556313985282/0/635563139852827086.jpg)

Finally, the last line executes whatever is in the "systemcommand" variable. If the variable is empty, then no command is executed.

[](https://img.wonderhowto.com/img/original/58/46/63556312393579/0/635563123935795846.jpg)

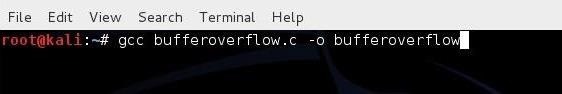
Finally, let's save this file as *bufferoverflow.c*.

Step 3Compile

The next step is to compile our new program. Compiling is the process of converting our source code, in this case C, into machine code. It's required whenever we write code in a compiled language, unlike say, Python, that is an interpreted language (interpreted languages are converted to machine language on the fly at run-time, line by line, and are thereby slower).

We need to use the GNU C Compiler (gcc) on the file bufferoverflow.c and output (-o) the compiled to a new file named "bufferoverflow" or whatever you choose to call it.

***kali> gcc bufferoverflow.c -o bufferoverflow***

[](https://img.wonderhowto.com/img/original/04/06/63556312409470/0/635563124094700406.jpg)

This may produce a few errors, but you can largely ignore those.

Step 4Run the Program

Now, let's run our little "bufferoverflow" program.

***kali > ./bufferoverflow***

[](https://img.wonderhowto.com/img/original/42/29/63556312420579/0/635563124205794229.jpg)

Notice that it first responded with the memory location of our variable "place," and then the memory location of our variable "systemcommand," and third it calculates that there are 16 bytes between these two memory locations. It then prompts the user for "the best place on the web to learn hacking" and, of course, the user responds "Null Byte." Finally, our little program responds with the obvious truth "The best place to learn hacking is Null Byte."

Step 5Overflow the Buffer

Now, let's run this program and try to overflow the memory area for the variable "place" into the memory area for "systemcommand." If we can overflow that memory area into the variable "systemcommand," we should be able to execute any system command on the system. For instance, we might be able to execute a command shell or display the contents of the */etc/shadow* file.

We know from the third printf statement, the space between the "place" variable and the "systemcommand" variable is 16 bytes. This means that if we enter more than 16 ASCII characters (each ASCII character is one byte) when prompted, starting with the 17th character, whatever we input will move into the next variable, in this case, "systemcommand."

Let's now enter the following when prompted:

***kali> nnnnnnnnnnnnnnnncat /etc/shadow***

When we do this, the first 16 characters will go into the "place" variable and the 17th character and everything after will overflow into the "systemcommand" variable. The final line of our code will then execute the system command variable.

[](https://img.wonderhowto.com/img/original/26/91/63556312433376/0/635563124333762691.jpg)

Notice that that we have been able to overflow the "place" variable into the "systemcommand" variable and have been able to "[cat](https://null-byte.wonderhowto.com/how-to/hack-like-pro-linux-basics-for-aspiring-hacker-part-3-managing-directories-files-0147293/)" the contents of the */etc/shadow* file thereby showing us all the users and their hashed passwords. Congratulations! You have successfully overflowed the buffer and run your own code.

Over the course of this year, 2015, we will work step by step toward developing our own zero-day exploit in [this series](https://null-byte.wonderhowto.com/how-to/exploit-building/), so keep coming back, my novice hackers!

Cover image via [Shutterstock](http://www.shutterstock.com/pic-113104090/stock-vector-black-bomb-isolated-on-white-background-vector-illustration.html" \t "_blank)