1. (2 points) Draw as good as possible figure of the following exploits and how they function:

- (a) off-by-one,
- (b) heap overflow and
- (c) function pointer.

You can again use as a bases the articles from the material section: ”Blended attacks...” and http://arstechnica.com/security/2015/08/how-security-flaws-work-the-buffer-overflow/

Buffer Overflow

First just a basic buffer overflow:

Code:

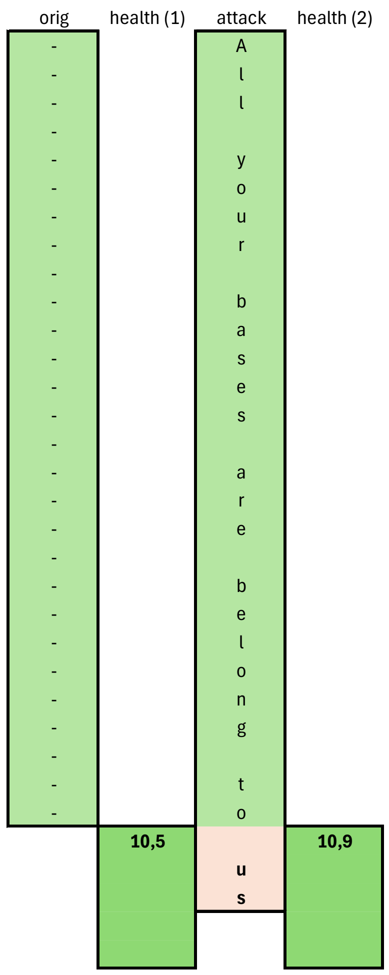
```
#include <string.h>
#include <stdio.h>

void overTurn(char *bar)
{
   float myHealth = 10.5;
   char  target[28];

   printf("My Health before = %f\n", myHealth);
   printf("Attack <%s>\n", bar);

   memcpy(target, bar, strlen(bar));  // no bounds checking...

   printf("My Health after overturn = %f\n", myHealth);
}

int main(int argc, char **argv)
{
   overTurn("All your bases are belong to us");
   overTurn("All your bases are belong to Us");
   return 0;
}
```

Output:

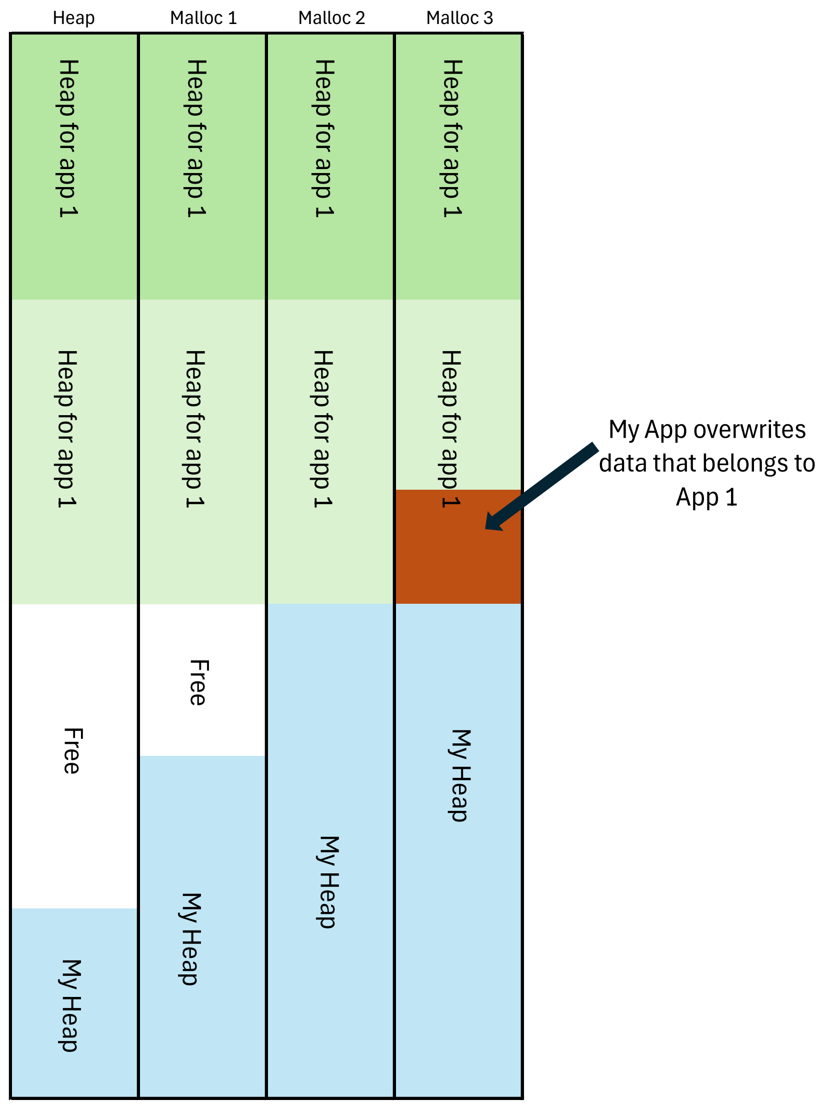
```
My Health before = 10.500000
Attack <All your bases are belong to us>
My Health after overturn = 15.216095
My Health before = 10.500000
```

Attack <All your bases are belong to Us>
My Health after overturn = 15.208282

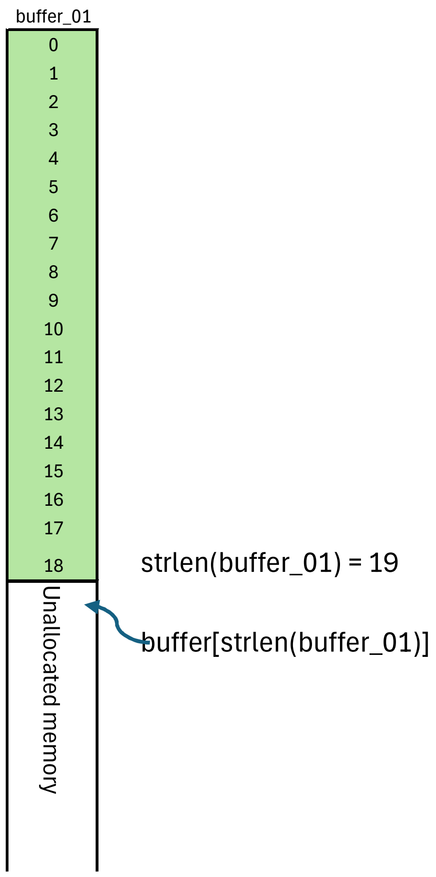
Memory Map:



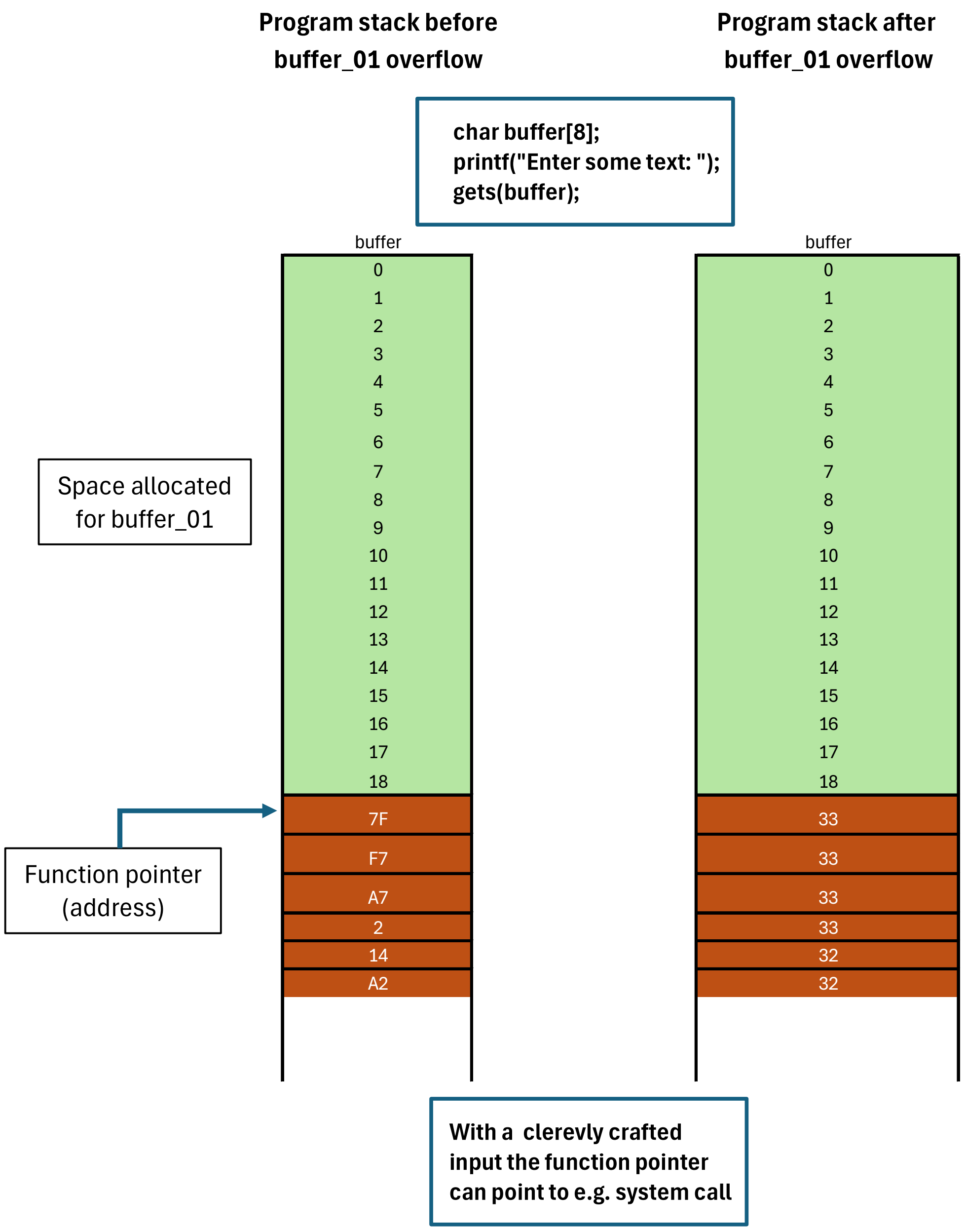
A - Heap overflow



B - Off-By-One


buffer_01
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
Unallocated memory
strlen(buffer_01) = 19
buffer[strlen(buffer_01)]

C - function pointer


Program stack before buffer_01 overflow
Program stack after buffer_01 overflow
char buffer[8];
printf("Enter some text: ");
gets(buffer);
buffer
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
7F
F7
A7
2
14
A2
buffer
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
33
33
33
33
32
32
Space allocated for buffer_01
Function pointer (address)
With a clerevly crafted input the function pointer can point to e.g. system call