USER

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

#define FLAGSIZE_MAX 64

char flag[FLAGSIZE_MAX];

void sigsegv_handler(int sig) {
  printf("%s\n", flag);
  fflush(stdout);
  exit(1);
}

void vuln(char *input){
  char buf2[16];
  strcpy(buf2, input);
}

int main(int argc, char **argv){

  FILE *f = fopen("flag.txt","r");
  if (f == NULL) {
    printf("%s %s", "Please create 'flag.txt' in this directory with your",
                    "own debugging flag.\n");
    exit(0);
  }

  fgets(flag,FLAGSIZE_MAX,f);
  signal(SIGSEGV, sigsegv_handler); // Set up signal handler

  gid_t gid = getegid();
  setresgid(gid, gid, gid);


  printf("Input: ");
  fflush(stdout);
  char buf1[100];
  gets(buf1);
  vuln(buf1);
  printf("The program will exit now\n");
  return 0;
```

}
Let's start off simple, can you overflow the correct buffer?

Input:

ASSISTANT

To overflow the correct buffer, we need to identify the vulnerable buffer in the code.

Identify Vulnerability:

The vulnerable buffer is `buf2` in the `vuln` function, which has a size of 16 bytes. However, the `strcpy` function is used to copy the input into `buf2` without any bounds checking, making it vulnerable to a buffer overflow attack.

Exploit Vulnerability:

To overflow the `buf2` buffer, we need to provide an input that is longer than 16 bytes. However, the `gets` function in the `main` function reads input into the `buf1` buffer, which has a size of 100 bytes.

Since the `gets` function does not perform any bounds checking, we can overflow the `buf1` buffer by providing an input that is longer than 100 bytes. However, this will not directly overflow the `buf2` buffer.

To overflow the `buf2` buffer, we need to provide an input that is longer than 16 bytes to the `vuln` function. Since the `vuln` function is called with the `buf1` buffer as an argument, we can overflow the `buf2` buffer by providing an input that is longer than 16 bytes to the `gets` function.

Here's an example input that can overflow the `buf2` buffer:

Input:
`AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AA

AAAA

AAAA

AAAA

AAAA