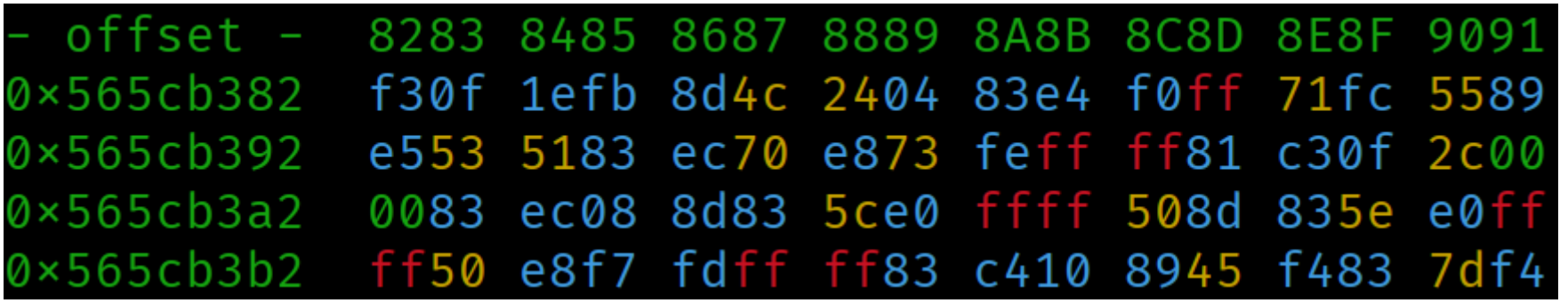
PicoCTF 2022 – Binary Overflow 0

```
- offset -  8283 8485 8687 8889 8A8B 8C8D 8E8F 9091
0×565cb382  f30f 1efb 8d4c 2404 83e4 f0ff 71fc 5589
0×565cb392  e553 5183 ec70 e873 feff ff81 c30f 2c00
0×565cb3a2  0083 ec08 8d83 5ce0 ffff 508d 835e e0ff
0×565cb3b2  ff50 e8f7 fdff ff83 c410 8945 f483 7df4
```

This challenge is an introduction to binary hacking memory vulnerabilities

Memory Vulnerabilities



Memory vulnerabilities are flaws in a program which allow the contents of the program's memory to be modified in an unintentional manner

Memory Vulnerabilities

Some consequences of memory vulnerabilities can be seen here

```
Memory Vuln Consequences

* Program Crashes
* Program Data Leaks
* Program Data Corruption
* Arbitrary Code Execution
```

Challenge Goal: Crash to Win

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

```
signal(SIGSEGV, sigsegv_handler);
```

The code for this binary indicates that if the program crashes, then contents of the `flag.txt` file will be revealed

Challenge Goal: Crash to Win

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

```
signal(SIGSEGV, sigsegv_handler);
```

`SIGSEGV` refers to a segmentation violation, which is a program memory-related crash

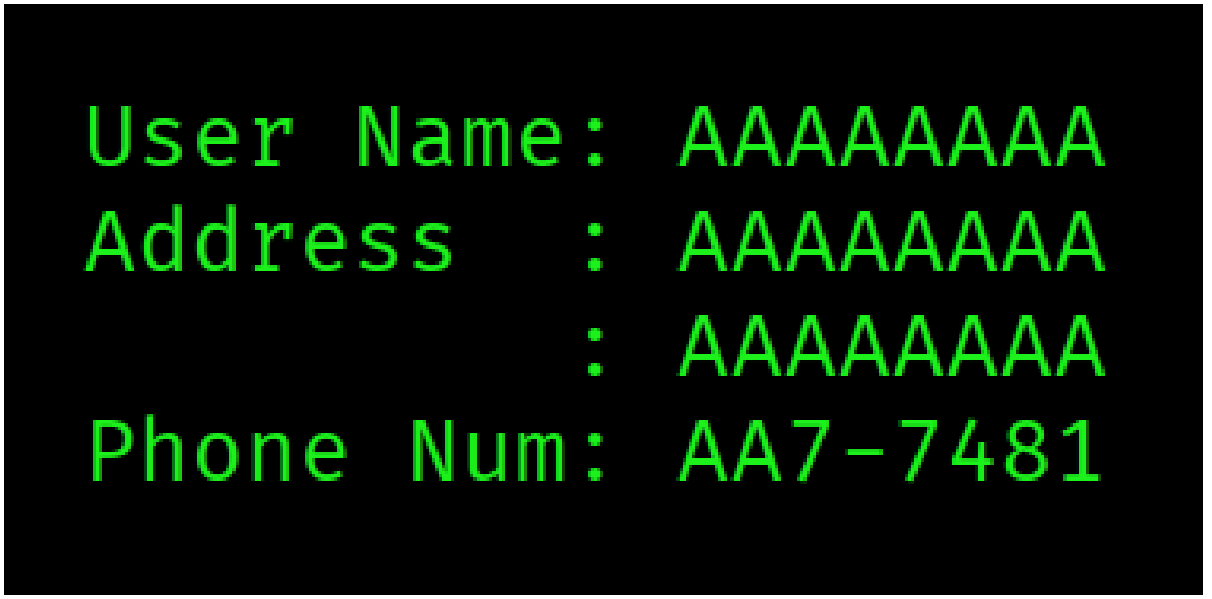
Challenge Goal: Crash to Win

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

```
signal(SIGSEGV, sigsegv_handler);
```

Segmentation violations happen when a program attempts to access invalid memory addresses. In this binary, it happens due to `buffer overflow`

Memory Vuln: Buffer Overflow



Buffer overflow is a program memory vulnerability where users are able to **overflow** a program's memory **buffer**, which results other parts of the program's memory to be overwritten

Memory Vuln: Buffer Overflow

```
char buf1[100];
gets(buf1);
```

In this C code, the string variable is created with a set maximum size, so the user can send up to 99 characters and write it to `buf1`

Memory Vuln: Buffer Overflow

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

```
vuln(buf1);
```

However, the program also takes buf1 and copies it to the buf2 array, which has a max size of 16 bytes

Memory Vuln: Buffer Overflow

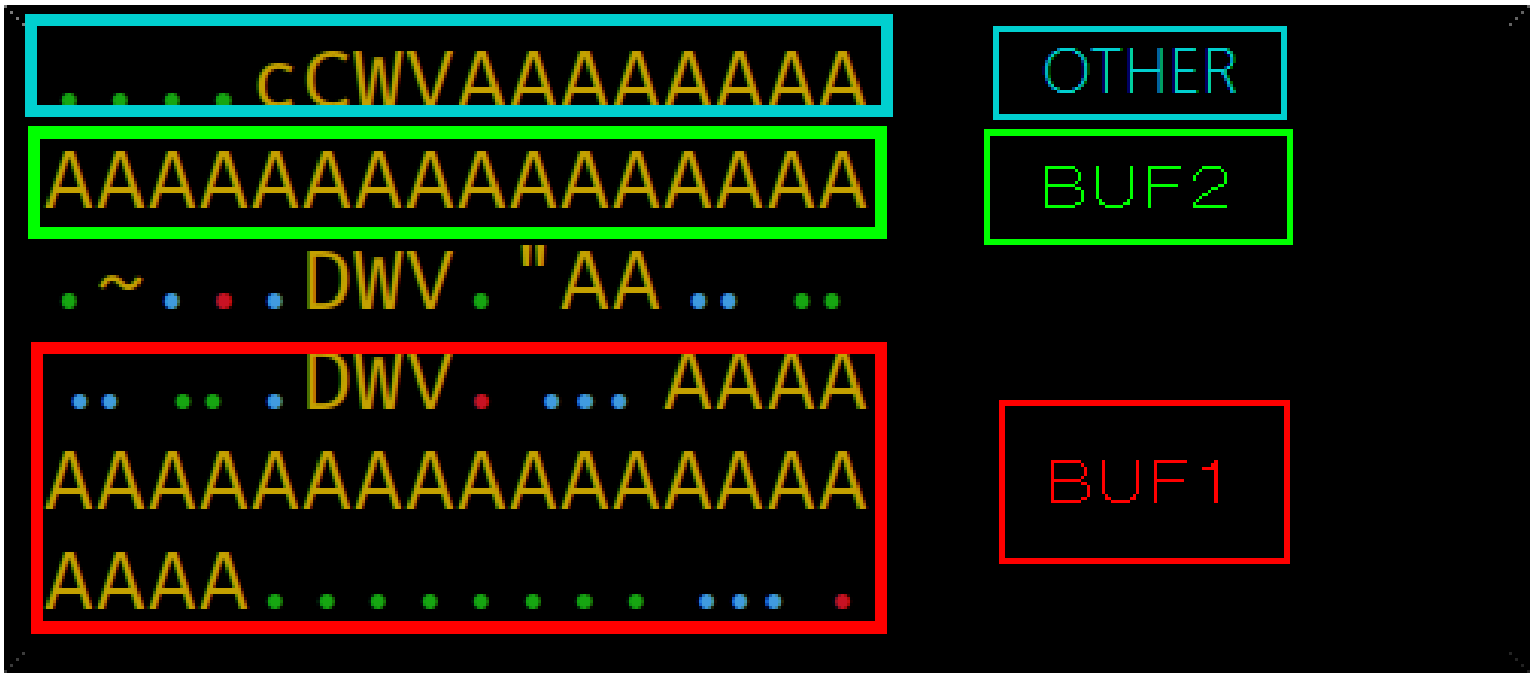
```
Vulnerable C Functions

gets    used to read a line of user input

strcpy  copies a string from
        one location to another
```

The reason why the buffer overflow can occur in this binary is because it was coded with memory unsafe functions, specifically `strcpy`, which does not validate the size of the strings

Memory Vuln: Buffer Overflow



So if we send more than 16 bytes to the user input, it'll be copied into the BUF1 buffer, with excess input overflowing into other memory addresses

Challenge Goal: Crash to Win

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

```
signal(SIGSEGV, sigsegv_handler);
```

This will crash the program due to segmentation violation, and this binary is programmed to output the flag in that case.