**Department of Computer Science and Engineering**
**Islamic University of Technology (IUT)**
A subsidiary organ of OIC

# SWE 4504 Software Security

**Lab Final**

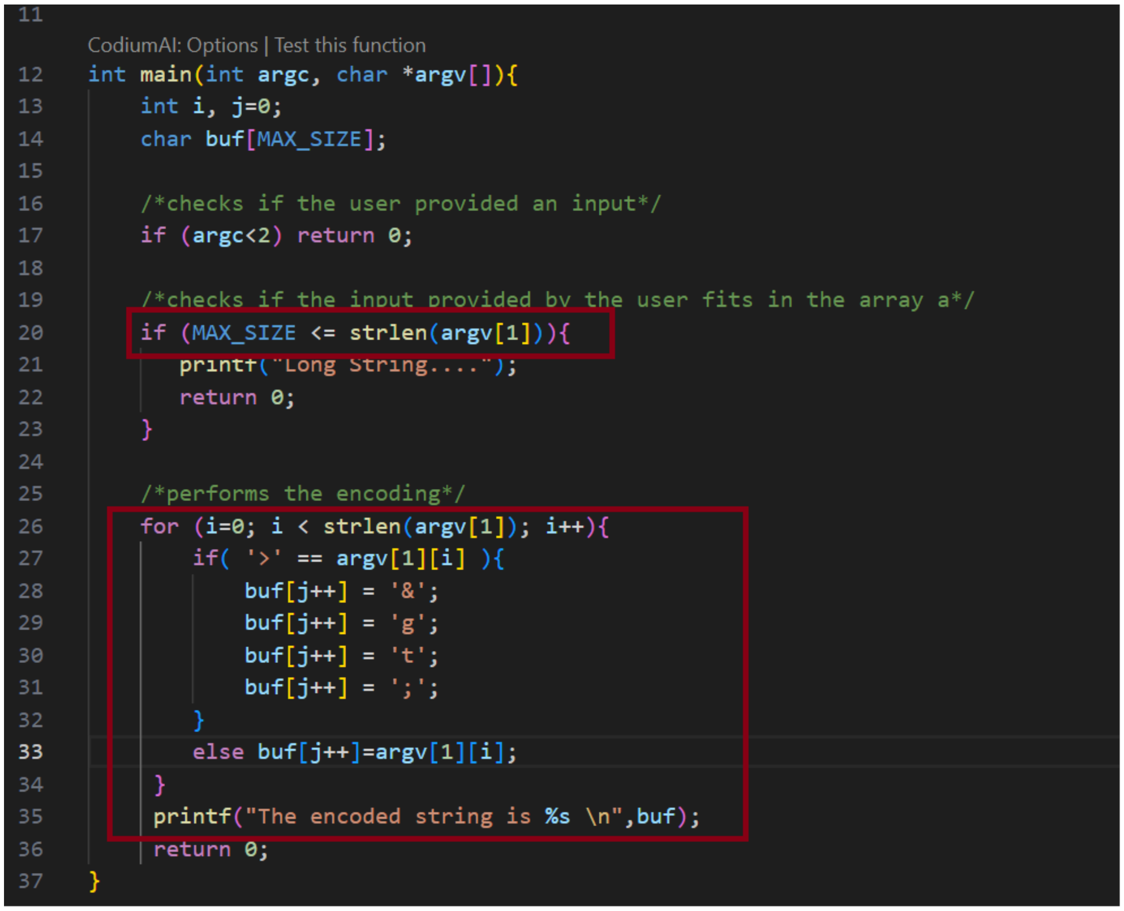
Team Members:

Mirza Mohammad Azwad, ID: 200042121

Programme: Software Engineering,
Dept. of Computer Science and Engineering,
Islamic University of Technology

# Program 2

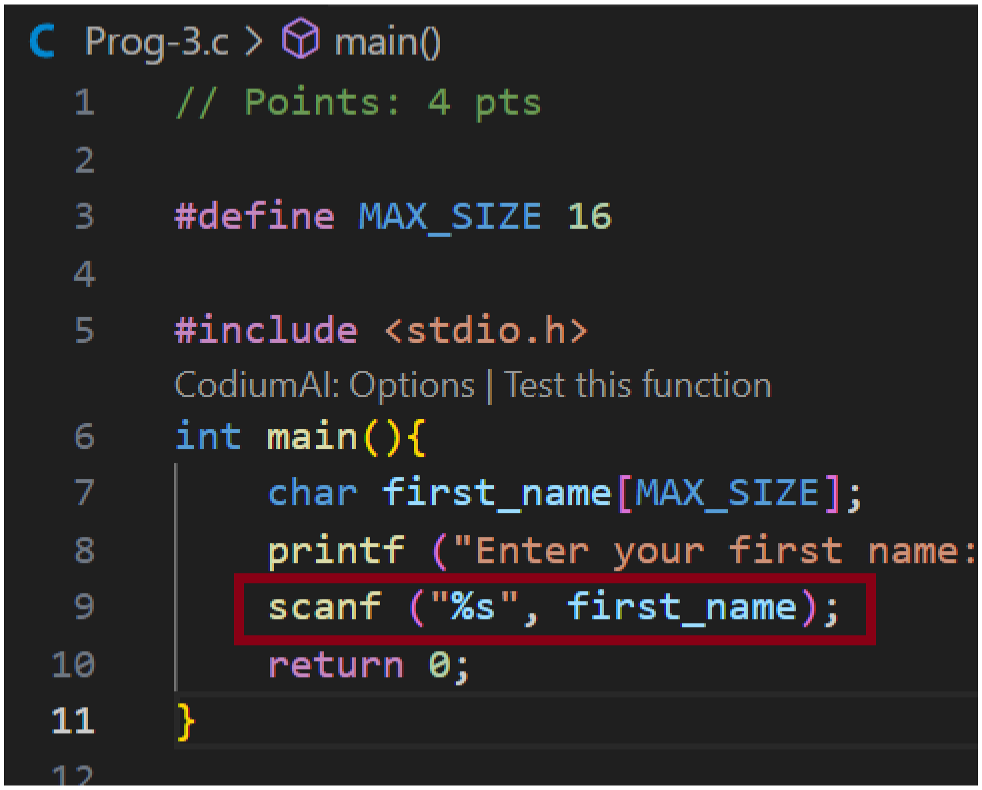
**Vulnerability: Improper Input Validation+Buffer Overflow**
**Location:** Line 26 onwards
**Cause:** While we check the maximum size of the argument one could still overflow the buffer using a number of “>” sign in succession as the garbage followed by so let’s say we have a payload of 16 bytes as dictated by MAX_SIZE. But the first 4 bytes is “>>>>” which essentially completes the range of buf size and then the remaining can be used to overflow buf, and potentially overwrite i or j, and we can also assign other values to j based on the memory location we would like to access other memory locations to dump content into.



```
11
12  int main(int argc, char *argv[]){
13      int i, j=0;
14      char buf[MAX_SIZE];
15
16      /*checks if the user provided an input*/
17      if (argc<2) return 0;
18
19      /*checks if the input provided by the user fits in the array a*/
20      if (MAX_SIZE <= strlen(argv[1])){
21         printf("Long String....");
22         return 0;
23      }
24
25      /*performs the encoding*/
26      for (i=0; i < strlen(argv[1]); i++){
27          if( '>' == argv[1][i] ){
28              buf[j++] = '&';
29              buf[j++] = 'g';
30              buf[j++] = 't';
31              buf[j++] = ';';
32          }
33          else buf[j++]=argv[1][i];
34       }
35       printf("The encoded string is %s \n",buf);
36       return 0;
37  }
```

# Program 3

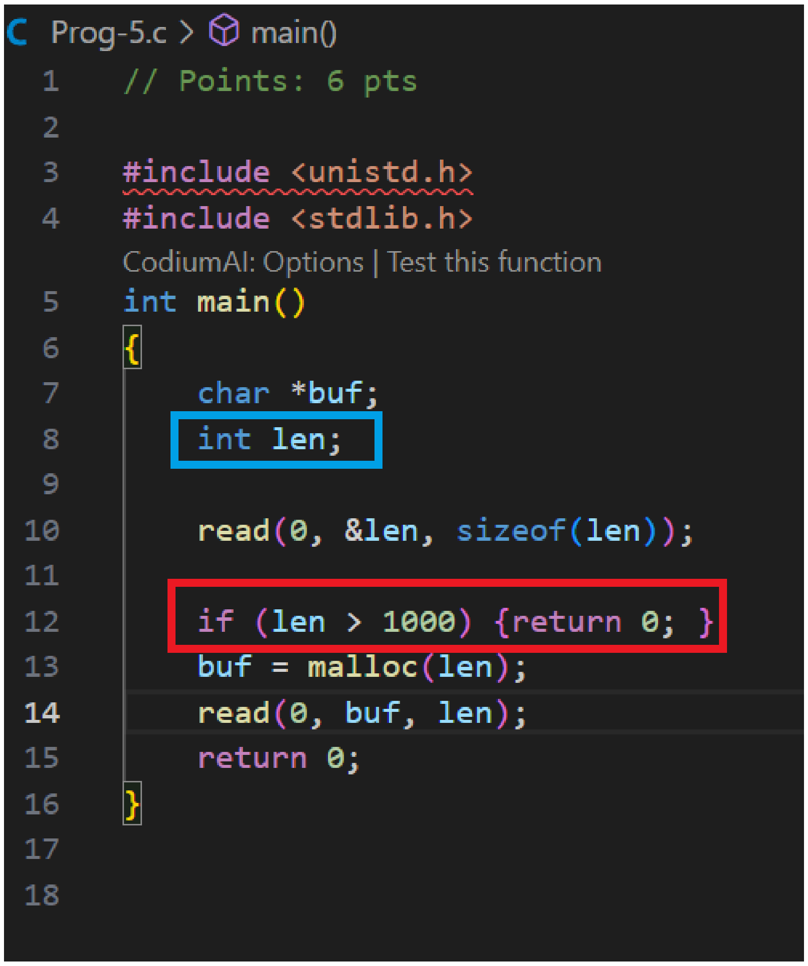
**Vulnerability**: **Buffer Overflow Vulnerability**
**Location**: Line 9(Marked in Red)
**Cause**: scanf function is vulnerable to overflow as it continues taking input until it reaches a '\n' or a ' '. This scanf function can be used to overflow with a garbage and we overflow the first_name treating as a buffer. In this case way we can overflow the return address of main. This means we could use this to essentially return to a SHELLCODE to execute it or some other kind of vulnerability.



```
Prog-3.c > main()
 1   // Points: 4 pts
 2
 3   #define MAX_SIZE 16
 4
 5   #include <stdio.h>
     CodiumAI: Options | Test this function
 6   int main(){
 7       char first_name[MAX_SIZE];
 8       printf ("Enter your first name:
 9       scanf ("%s", first_name);
10       return 0;
11   }
12
```

## Program 5

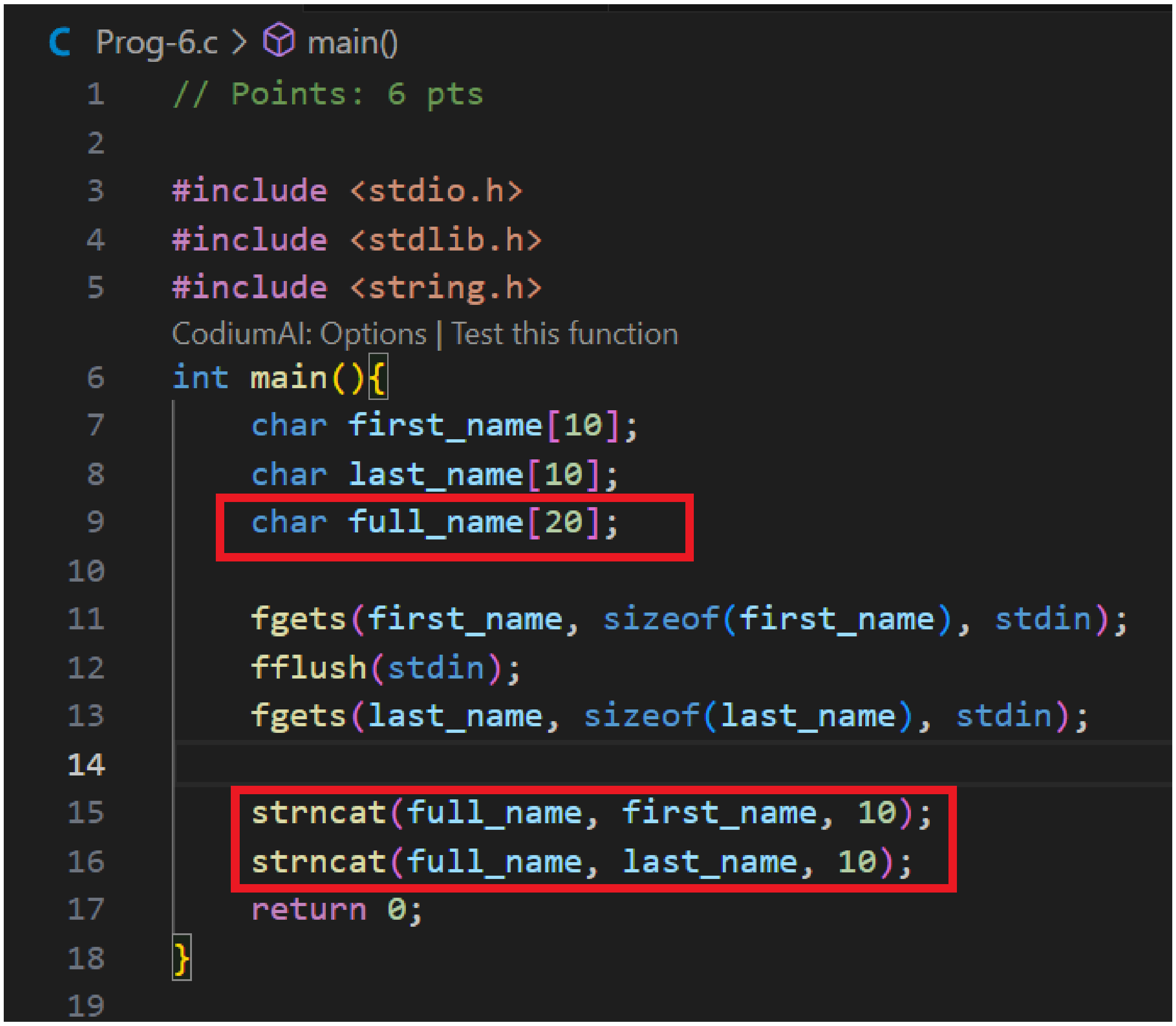
**Vulnerability: Improper Input Validation**
**Location:** In line 12(Marked in Red)
**Cause**: we check for whether or not the length exceeds a given range but what if someone enters the length(Marked in Blue) as negative, that could potentially be a memory enlargement hack in a sense, since malloc(size_t ss) is basically the definition and size_t is unsigned so essentially entering -1 would imply maximum unsigned number, so we can actually enter a larger string than 1000 potentially.



```
 1  // Points: 6 pts
 2
 3  #include <unistd.h>
 4  #include <stdlib.h>
 5  int main()
 6  {
 7      char *buf;
 8      int len;
 9
10      read(0, &len, sizeof(len));
11
12      if (len > 1000) {return 0; }
13      buf = malloc(len);
14      read(0, buf, len);
15      return 0;
16  }
17
18
```

# Program 6

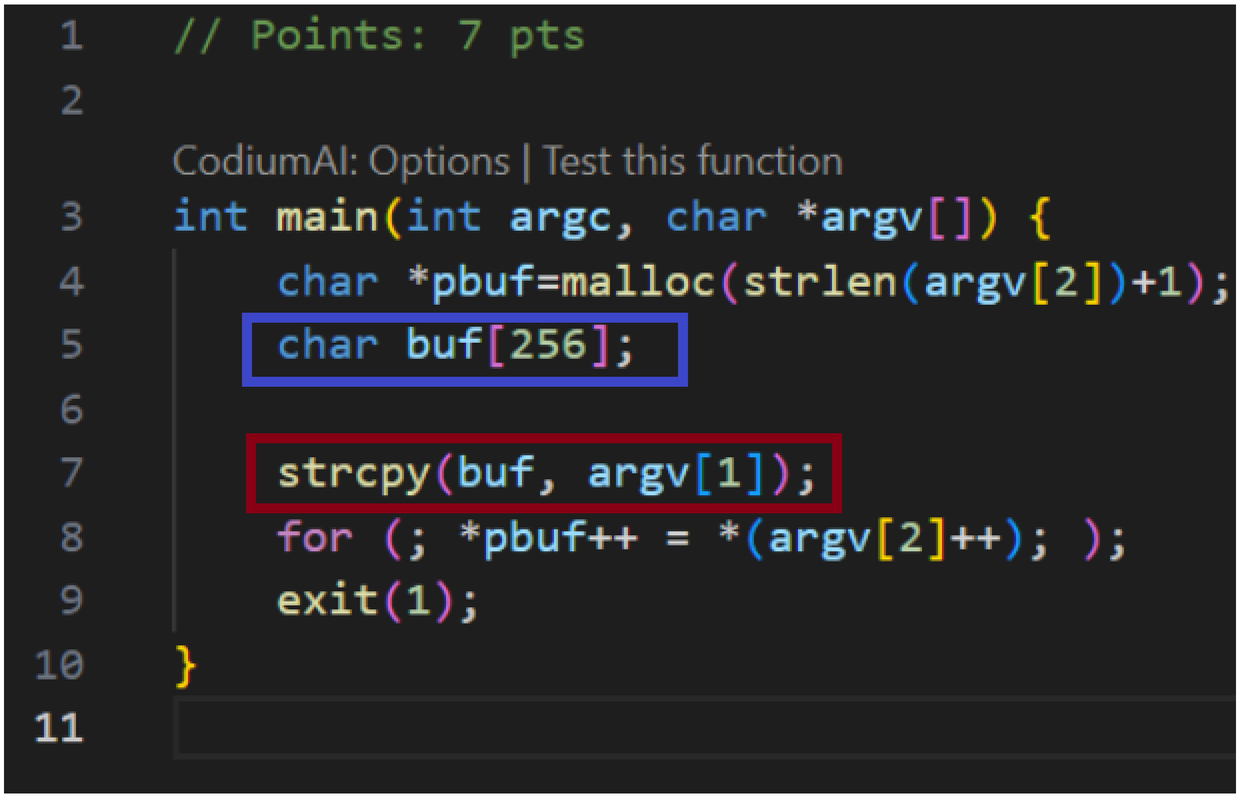
**Vulnerability: Improper Variable Initialization**
**Location:** Line 9, 15, 16
**Cause:** The full_name could potentially have garbage values assigned and might not be null to begin with. The presence of garbage values may not cause the desired full name assigned to be assigned and rather an incorrect full_name could end up being assigned



```c
1   // Points: 6 pts
2
3   #include <stdio.h>
4   #include <stdlib.h>
5   #include <string.h>
6   int main(){
7       char first_name[10];
8       char last_name[10];
9       char full_name[20];
10
11      fgets(first_name, sizeof(first_name), stdin);
12      fflush(stdin);
13      fgets(last_name, sizeof(last_name), stdin);
14
15      strncat(full_name, first_name, 10);
16      strncat(full_name, last_name, 10);
17      return 0;
18  }
19
```

# Program 8

Vulnerability: **Buffer Overflow Vulnerability and Pointer Dereferencing Vulnerability**
Location: Line 7 (Marked Red),
Cause: strcpy function is vulnerable to overflow, and the buf is what can be overflown. Since we are passing arguments to the code before execution from the terminal,. The arguments can be manipulated with argv[1] to get a large string that exceeds buf. This means we can change the value of pbuf to point to a different string in a known memory location, provided that we have around 256 bytes of garbage followed by a known memory location as a 32-bit address. Also, we could add an equal length garbage as part of our argv [2]. Equal in the sense that argv[2] and the vulnerable string we want to access are of equal length.



```
1    // Points: 7 pts
2
     CodiumAI: Options | Test this function
3    int main(int argc, char *argv[]) {
4        char *pbuf=malloc(strlen(argv[2])+1);
5        char buf[256];
6
7        strcpy(buf, argv[1]);
8        for (; *pbuf++ = *(argv[2]++); );
9        exit(1);
10   }
11
```