Code vulnerability analysis

Func1:-

```
1  void func1()
2  {
3      char * data;
4      char * dataBuffer = (char *)ALLOCA(100*sizeof(char));
5      memset(dataBuffer, 'A', 100-1);
6      dataBuffer[100-1] = '\0';
7      data = dataBuffer - 8;
8      {
9          char source[100];
10         memset(source, 'C', 100-1);
11         source[100-1] = '\0';
12         strcpy(data, source);
13         if(data != NULL)
14         {
15             printf("%s\n", data);
16         }
17     }
18 }
```

Vulnerability:-

Line number	Vulnerability
4	Should define ALLOCA as macro and use of alloca() is discourage in low-level programming because when it fails to allocate memory it throws StackOverflow error, instead use malloc if you want to allocate memory at runtime.
5	Should check if memory is allocated or not before assigning data
12	Might cause invalid memory access.

Func2:-

```
1  void func2()
2  {
3      char * data;
4      data = NULL;
5      data = (char *)calloc(100, sizeof(char));
6      strcpy(data, "A String");
7      if(data != NULL)
8      {
9          printf("%s\n", data);
10     }
11 }
```

Vulnerability:-

Line number	**Vulnerability**
6	Check if memory is assigned or not before assigning data.

Func3:-

```
1  void func3()
2  {
3      char * password;
4      char passwordBuffer[100] = "";
5      password = passwordBuffer;
6      strcpy(password, PASSWORD);
7      {
8          HANDLE pHandle;
9          char * username = "User";
10         char * domain = "Domain";
11         /* Let's say LogonUserA is a custon authentication function*/
12         if (LogonUserA(
13                     username,
14                     domain,
15                     password,
16                     &pHandle) != 0)
17         {
18             printf("User logged in successfully.\n");
19             CloseHandle(pHandle);
20         }
21         else
22         {
23             printf("Unable to login.\n");
24         }
25     }
26 }
```

Vulnerability:-

Line number	**Vulnerability**
6	PASSWORD is stored in code section of memory because it is macro, if any attacker get access to binary of the program, he/she can access the password
16	Someone can tamper LogonUserA function to change password at runtime as we are passing address of password which is allocated in stack and someone can change content of memory of stack.

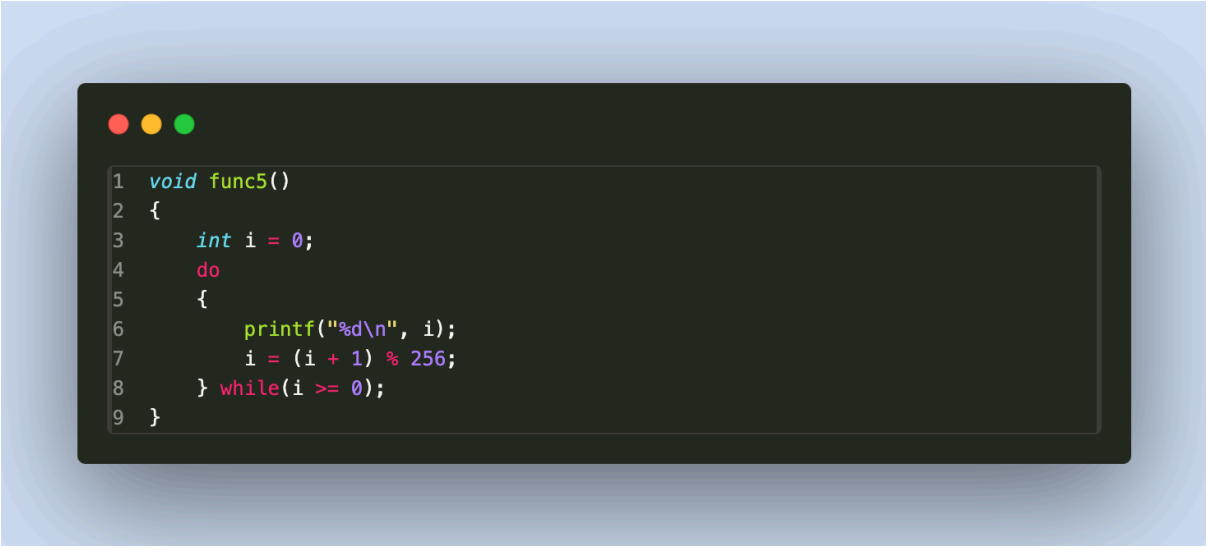
Func4:-

```
1  static void func4()
2  {
3      char * data;
4      data = NULL;
5      data = (char *)calloc(20, sizeof(char));
6      if (data != NULL)
7      {
8          strcpy(data, "Initialize");
9          if(data != NULL)
10         {
11             printf("%s\n", data);
12         }
13         free(data);
14     }
15 }
```

Vulnerability:-

Line number	Vulnerability
8	If strcpy fails then we will not be able to free memory leads ot memory leak

Func5:-



```c
void func5()
{
    int i = 0;
    do
    {
        printf("%d\n", i);
        i = (i + 1) % 256;
    } while(i >= 0);
}
```

Vulnerability:-

Line number	Vulnerability
4-8	Infinite loop

Func6:-

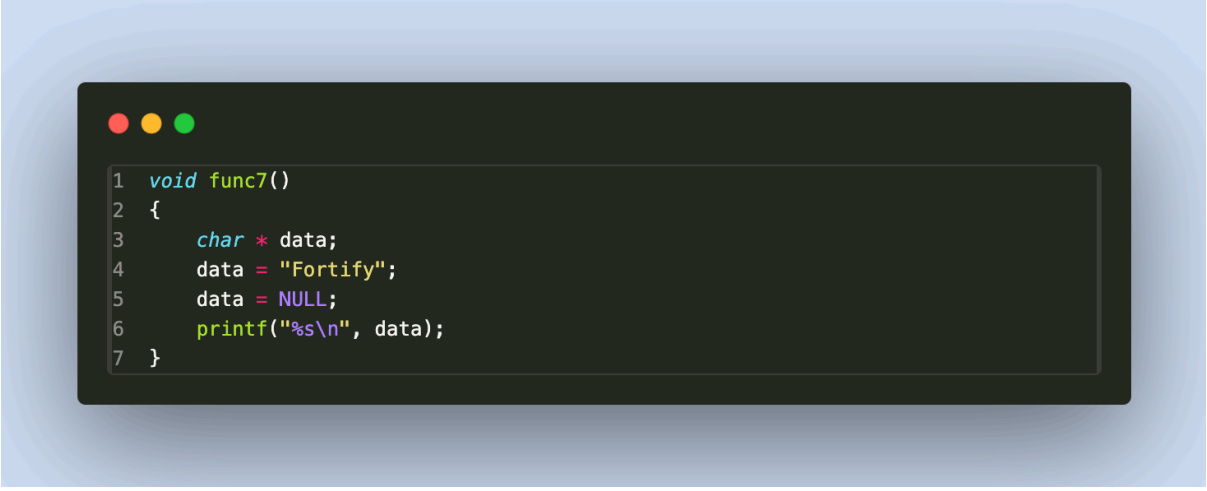
```c
void func6()
{
    char dataBuffer[100] = "";
    char * data = dataBuffer;
    printf("Please enter a string: ");
    if (fgets(data, 100, stdin) < 0)
    {
        printf("fgets failed!\n");
        exit(1);
    }
    if(data != NULL)
    {
        printf("%s\n", data);
    }

}
```

Vulnerability:-

Line number	Vulnerability
6	It won’t catch error as fgest returns NULL when fail to read and NULL is 0

Func7:-



```
1  void func7()
2  {
3      char * data;
4      data = "Fortify";
5      data = NULL;
6      printf("%s\n", data);
7  }
```

Vulnerability:-

Line Number	Vulnerability
6	Segmentation fault as trying to read memory at NULL address.