Lab 3 - Memory Safety

Task 1 - Getting Started

1. Install GCC and Valgrind

- **GCC** was already installed on my device. I verified this using the command:

```
gcc --version
```

- **Valgrind** was installed using the package manager:

```
sudo apt install valgrind
valgrind --version
```

```
mohamad@mohamad-HP-ProBook-430-G7:~$ gcc
gcc: fatal error: no input files
compilation terminated.
mohamad@mohamad-HP-ProBook-430-G7:~$ gcc --version
gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

mohamad@mohamad-HP-ProBook-430-G7:~$ valgrind
Command 'valgrind' not found, but can be installed with:
sudo snap install valgrind  # version 3.24.0, or
sudo apt  install valgrind  # version 1:3.18.1-1ubuntu2
See 'snap info valgrind' for additional versions.
mohamad@mohamad-HP-ProBook-430-G7:~$ sudo apt  install valgrind
[sudo] password for mohamad:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libc6-i386
Suggested packages:
  valgrind-dbg valgrind-mpi kcachegrind alleyoop valkyrie
The following NEW packages will be installed:
  libc6-i386 valgrind
0 upgraded, 2 newly installed, 0 to remove and 1 not upgraded.
Need to get 16,9 MB of archives.
After this operation, 91,8 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://ru.archive.ubuntu.com/ubuntu jammy-updates/main amd64 libc6-i386 amd64 2.35-0ubuntu3.9 [2 838 kB]
Get:2 http://ru.archive.ubuntu.com/ubuntu jammy/main amd64 valgrind amd64 1:3.18.1-1ubuntu2 [14,1 MB]
Fetched 16,9 MB in 3s (5 548 kB/s)
Selecting previously unselected package libc6-i386.
(Reading database ... 227198 files and directories currently installed.)
Preparing to unpack .../libc6-i386_2.35-0ubuntu3.9_amd64.deb ...
Unpacking libc6-i386 (2.35-0ubuntu3.9) ...
Selecting previously unselected package valgrind.
Preparing to unpack .../valgrind_1%3a3.18.1-1ubuntu2_amd64.deb ...
Unpacking valgrind (1:3.18.1-1ubuntu2) ...
Setting up libc6-i386 (2.35-0ubuntu3.9) ...
Setting up valgrind (1:3.18.1-1ubuntu2) ...
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for libc-bin (2.35-0ubuntu3.9) ...
mohamad@mohamad-HP-ProBook-430-G7:~$ valgrind --version
valgrind-3.18.1
mohamad@mohamad-HP-ProBook-430-G7:~$
```

2. Create `program1.c`

The content of `program1.c` is as follows:

```c
#include<stdio.h>
#include<stdlib.h>

void program1(int N) {
    int *arr = malloc(N);
    for(int i = 0; i < N; i++) {
        arr[i] = i * i;
        printf("arr[%d] = %d\n", i, arr[i]);
    }
}

int main() {
    program1(4); // Should print the array [0, 1, 4, 9]
}
```

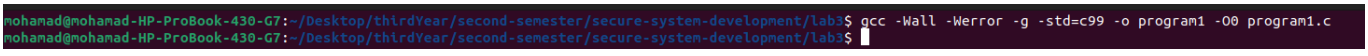
3. Compile `program1.c`

The compilation command used is:

```
gcc -Wall -Werror -g -std=c99 -o program1 -O0 program1.c
```

Explanation of Flags:

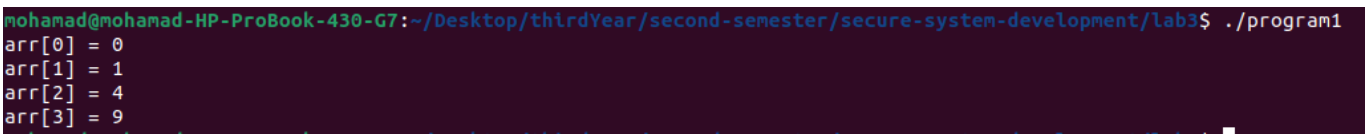
- `-Wall`: Enables all warnings.
- `-Werror`: Treats all warnings as errors.
- `-g`: Adds debugging information.
- `-std=c99`: Uses the C99 standard.
- `-o program1`: Specifies the output file name.
- `-O0`: Disables optimizations.



4. Run the Program

The program was executed using:

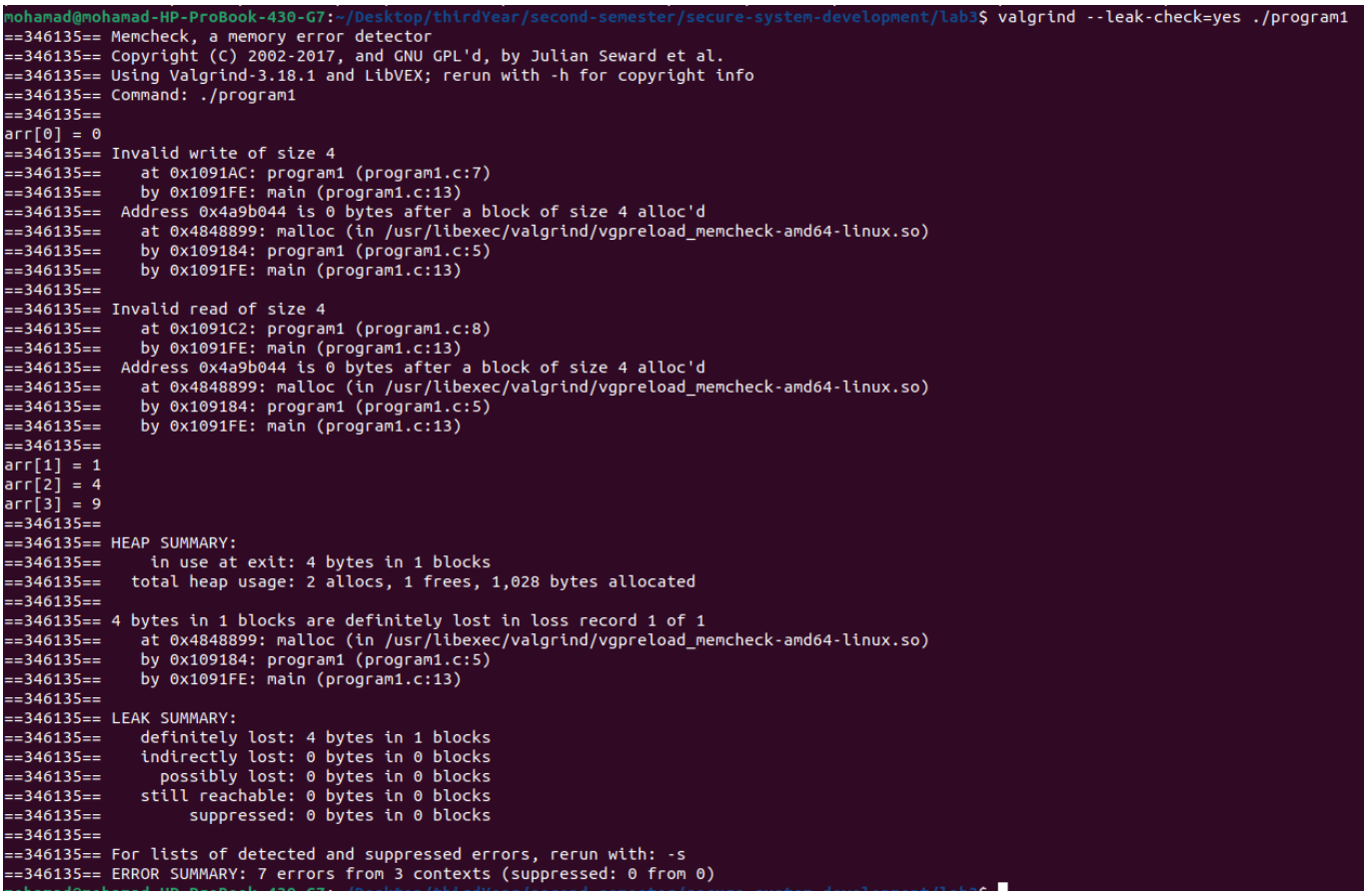
```
./program1
```



5. Run the Program with Valgrind

The program was executed with Valgrind using:

```
valgrind --leak-check=yes ./program1
```



6. Analyze Valgrind Output

The Valgrind output indicates two main memory-related issues:

1. **Invalid Memory Access (Out-of-Bounds Write & Read)**
    - The program attempts to write and read past the allocated memory (`0x4a9b044 is 0 bytes after a block of size 4 alloc'd`).
    - The allocated size (4 bytes) suggests an `int` array of size **1** (`malloc(4)`), but the program accesses multiple elements (`arr[1]`, `arr[2]`, `arr[3]`).

    CWE-787: Out-of-Bounds Write – MITRE Reference
    CWE-125: Out-of-Bounds Read – MITRE Reference

2. **Memory Leak**
    - The program allocates memory but does not free it (`definitely lost: 4 bytes`).
    - Every `malloc` should have a corresponding `free`.

    CWE-401: Memory Leak – MITRE Reference

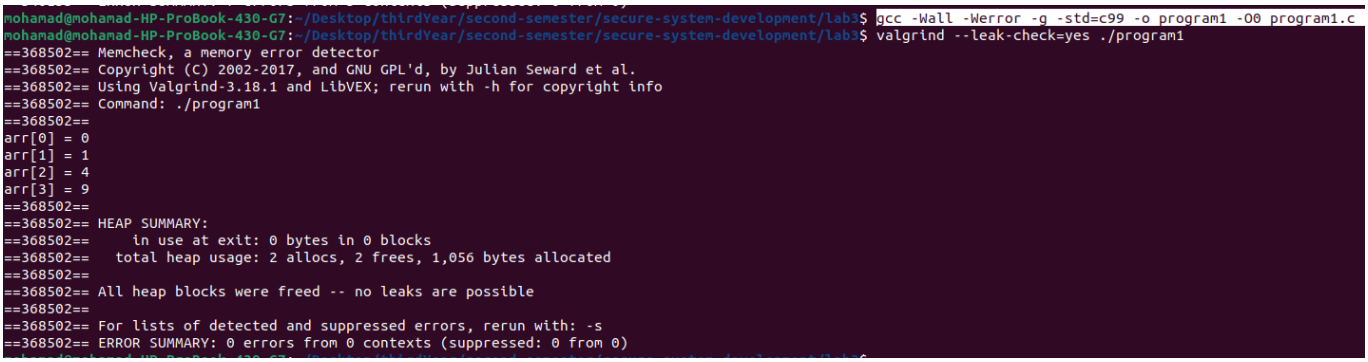
7. Propose Fixes

- **Fix the Memory Leak:** Add a `free` call after the `malloc` call.
- **Fix the Out-of-Bounds Access:** Ensure the array index is within the allocated size.

8. Verify Fixes

After applying the fixes, the program was recompiled and run with Valgrind:

```
gcc -Wall -Werror -g -std=c99 -o program1 -O0 program1.c
valgrind --leak-check=yes ./program1
```



Updated program1.c

Task 2 - More Programs

Program 2

1. Create `program2.c`

The content of `program2.c` is as follows:

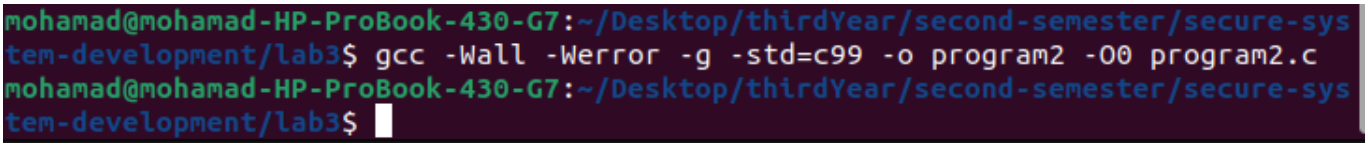
```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

void work(int* arr, unsigned N) {
    for(int i=1; i<N; i++) {
        arr[i] = arr[i-1] * 2;
    }
    free(arr);
}

void program2(unsigned N) {
    int* arr = (int*)malloc(N * sizeof(*arr));
    memset(arr, 0, sizeof(*arr));
    arr[0] = 1;
    work(arr, N);
    for(int i=0; i<N; i++) {
        printf("arr[%d] = %d\n", i, arr[i]);
```

```
    }
}

int main() {
    program2(4); // Should print the array [1, 2, 4, 8]
}
```

2. Compile with the following command `program2.c`

```
gcc -Wall -Werror -g -std=c99 -o program2 -O0 program2.c
```

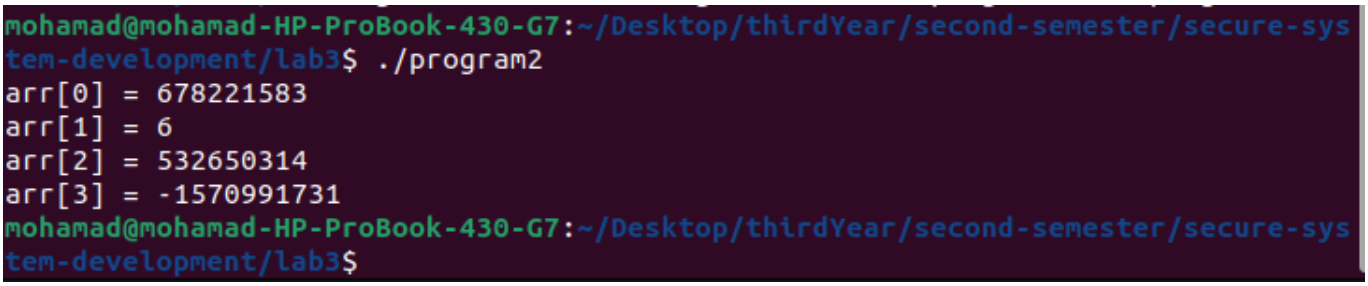
The program was compiled and run with Valgrind:



3. Run the program with the following command `program2.c`

```
./program2
```

- we running it, it doesn't give the correct ouptut



4. Run the program again with valgrind `program2.c`

```
valgrind --leak-check=yes ./program2
```

```
mohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-system-development/lab3$ valgrind --leak-check=yes ./program2
==15797== Memcheck, a memory error detector
==15797== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==15797== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==15797== Command: ./program2
==15797==
==15797== Invalid read of size 4
==15797==    at 0x10927E: program2 (program2.c:18)
==15797==    by 0x1092BA: main (program2.c:23)
==15797==  Address 0x4a9b040 is 0 bytes inside a block of size 16 free'd
==15797==    at 0x484B27F: free (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==15797==    by 0x10920A: work (program2.c:9)
==15797==    by 0x109260: program2 (program2.c:16)
==15797==    by 0x1092BA: main (program2.c:23)
==15797==  Block was alloc'd at
==15797==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==15797==    by 0x10922B: program2 (program2.c:13)
==15797==    by 0x1092BA: main (program2.c:23)
==15797==
arr[0] = 1
arr[1] = 2
arr[2] = 4
arr[3] = 8
==15797==
==15797== HEAP SUMMARY:
==15797==     in use at exit: 0 bytes in 0 blocks
==15797==   total heap usage: 2 allocs, 2 frees, 1,040 bytes allocated
==15797==
==15797== All heap blocks were freed -- no leaks are possible
==15797==
==15797== For lists of detected and suppressed errors, rerun with: -s
```

5. Analyze Valgrind Output `program2.c`

1. **Use-After-Free**
    - The program accesses memory (`arr[0]`, etc.) after it has been freed (`Address 0x4a9b040 is 0 bytes inside a block of size 16 free'd`).

CWE-416: Use After Free – MITRE Reference

6. Propose Fixes `program2.c`

- **Fix:** Move the `free` call after the loop that prints the array.

7. Verify Fixes `program2.c`

After applying the fixes, the program was recompiled and run with Valgrind:

```
gcc -Wall -Werror -g -std=c99 -o program2 -O0 program2.c
./program2
valgrind --leak-check=yes ./program2
```

```
mohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-system-development/lab3$ gcc -Wall -Werror -g -std=c99 -o program2 -O0 program2.c
mohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-system-development/lab3$ ./program2
arr[0] = 1
arr[1] = 2
arr[2] = 4
arr[3] = 8
mohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-system-development/lab3$ valgrind --leak-check=yes ./program2
==23382== Memcheck, a memory error detector
==23382== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==23382== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==23382== Command: ./program2
==23382==
arr[0] = 1
arr[1] = 2
arr[2] = 4
arr[3] = 8
==23382==
==23382== HEAP SUMMARY:
==23382==     in use at exit: 0 bytes in 0 blocks
==23382==   total heap usage: 2 allocs, 2 frees, 1,040 bytes allocated
==23382==
==23382== All heap blocks were freed -- no leaks are possible
==23382==
==23382== For lists of detected and suppressed errors, rerun with: -s
==23382== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
mohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-system-development/lab3$
```

8. Different output with Valgring and without it

- Without Valgrind, the OS may reuse freed memory, leading to incorrect output. Valgrind delays memory reuse, so values may still appear correct. The issue is **use-after-free**, causing undefined behavior. The fix is to free memory **after** using it.

Updated program2.c

Program 3

1. Create `program3.c`

The content of `program3.c` is as follows:

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

void* program3(unsigned N) {
    void *arr = malloc(N * sizeof(*arr));
    if((N < 1) || (arr = NULL)) {
        printf("%s\n", "Memory allocation falied!");
        return NULL;
    }
    printf("%s\n", "Memory allocation success!");
    return arr;
}

int main() {
    int* arr = (int*)program3(4); // Should typically succeed
    free(arr);
}
```

2. Compile with the following command `program3.c`

```
gcc -Wall -Werror -g -std=c99 -o program3 -O0 program3.c
```

The program was compiled and run with Valgrind:

3. Run the program with the following command `program3.c`

```
./program3
```

- we running it

```
mohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-systen-development/lab3$ ./program3
Memory allocation success!
mohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-systen-development/lab3$
```

4. Run the program again with valgrind `program3.c`

```
valgrind --leak-check=yes ./program3
```

```
mohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-system-development/lab3$ valgrind --leak-check=yes ./program3
==94731== Memcheck, a memory error detector
==94731== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==94731== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==94731== Command: ./program3
==94731==
Memory allocation success!
==94731==
==94731== HEAP SUMMARY:
==94731==     in use at exit: 4 bytes in 1 blocks
==94731==   total heap usage: 2 allocs, 1 frees, 1,028 bytes allocated
==94731==
==94731== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==94731==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==94731==    by 0x1091A2: program3 (program3.c:6)
==94731==    by 0x1091FC: main (program3.c:16)
==94731==
==94731== LEAK SUMMARY:
==94731==    definitely lost: 4 bytes in 1 blocks
==94731==    indirectly lost: 0 bytes in 0 blocks
==94731==      possibly lost: 0 bytes in 0 blocks
==94731==    still reachable: 0 bytes in 0 blocks
==94731==         suppressed: 0 bytes in 0 blocks
==94731==
==94731== For lists of detected and suppressed errors, rerun with: -s
==94731== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

5. Analyze Valgrind Output `program3.c`

1. **Memory Leak**
    - The program allocates memory but does not free it (`definitely lost: 4 bytes`).

   CWE-401: Memory Leak – MITRE Reference

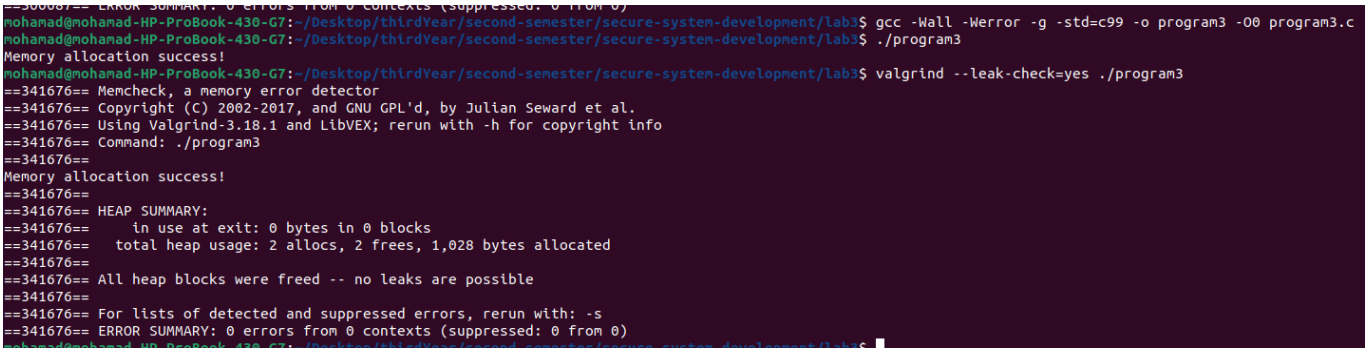
6. Propose Fixes `program3.c`

- **Fix:** Correct the condition in the `if` statement to check for `arr == NULL` instead of `arr = NULL`.

7. Verify Fixes `program3.c`

After applying the fixes, the program was recompiled and run with Valgrind:

```
gcc -Wall -Werror -g -std=c99 -o program3 -O0 program3.c
./program3
valgrind --leak-check=yes ./program3
```



Updated program3.c

Program 4

1. Create `program4.c`

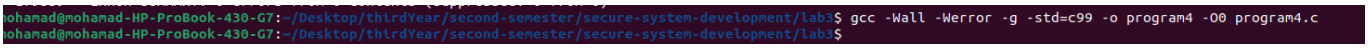
The content of `program4.c` is as follows:

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

char* getString() {
    char message[100] = "Hello World!";
    char* ret = message;
    return ret;
}

void program4() {
    printf("String: %s\n", getString());
}

int main() {
    program4();
}
```

2. Compile with the following command `program4.c`

```
gcc -Wall -Werror -g -std=c99 -o program4 -O0 program4.c
```

The program was compiled and run with Valgrind:



3. Run the program with the following command `program4.c`

```
./program4
```

- we running it

```
String: Hello World!
mohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-system-development/lab3$
```

4. Run the program again with valgrind `program4.c`

```
valgrind --leak-check=yes ./program4
```

```
mohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-system-development/lab3$ valgrind --leak-check=yes ./program4
==181176== Memcheck, a memory error detector
==181176== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==181176== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==181176== Command: ./program4
==181176==
==181176== Conditional jump or move depends on uninitialised value(s)
==181176==    at 0x484ED19: strlen (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==181176==    by 0x48E5D30: __vfprintf_internal (vfprintf-internal.c:1517)
==181176==    by 0x48CF79E: printf (printf.c:33)
==181176==    by 0x10923C: program4 (program4.c:12)
==181176==    by 0x109251: main (program4.c:16)
==181176==
==181176== Conditional jump or move depends on uninitialised value(s)
==181176==    at 0x484ED28: strlen (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==181176==    by 0x48E5D30: __vfprintf_internal (vfprintf-internal.c:1517)
==181176==    by 0x48CF79E: printf (printf.c:33)
==181176==    by 0x10923C: program4 (program4.c:12)
==181176==    by 0x109251: main (program4.c:16)
==181176==
==181176== Conditional jump or move depends on uninitialised value(s)
==181176==    at 0x48FA737: _IO_new_file_xsputn (fileops.c:1218)
==181176==    by 0x48FA737: _IO_file_xsputn@@GLIBC_2.2.5 (fileops.c:1196)
==181176==    by 0x48E600B: outstring_func (vfprintf-internal.c:239)
==181176==    by 0x48E600B: __vfprintf_internal (vfprintf-internal.c:1517)
==181176==    by 0x48CF79E: printf (printf.c:33)
==181176==    by 0x10923C: program4 (program4.c:12)
==181176==    by 0x109251: main (program4.c:16)
==181176==
==181176== Syscall param write(buf) points to uninitialised byte(s)
==181176==    at 0x4983887: write (write.c:26)
==181176==    by 0x48F9EEC: _IO_file_write@@GLIBC_2.2.5 (fileops.c:1180)
==181176==    by 0x48FB9E0: new_do_write (fileops.c:448)
==181176==    by 0x48FB9E0: _IO_new_do_write (fileops.c:425)
==181176==    by 0x48FB9E0: _IO_do_write@@GLIBC_2.2.5 (fileops.c:422)
==181176==    by 0x48FA6D4: _IO_new_file_xsputn (fileops.c:1243)
==181176==    by 0x48FA6D4: _IO_file_xsputn@@GLIBC_2.2.5 (fileops.c:1196)
==181176==    by 0x48E4FC9: outstring_func (vfprintf-internal.c:239)
==181176==    by 0x48E4FC9: __vfprintf_internal (vfprintf-internal.c:1593)
==181176==    by 0x48CF79E: printf (printf.c:33)
==181176==    by 0x10923C: program4 (program4.c:12)
==181176==    by 0x109251: main (program4.c:16)
==181176==  Address 0x4a9b048 is 8 bytes inside a block of size 1,024 alloc'd
==181176==    at 0x4848899: malloc (in /usr/libexec/valgrind/vgpreload_memcheck-amd64-linux.so)
==181176==    by 0x48EDBA3: _IO_file_doallocate (filedoalloc.c:101)
==181176==    by 0x48FCCDF: _IO_doallocbuf (genops.c:347)
==181176==    by 0x48FBF5F: _IO_file_overflow@@GLIBC_2.2.5 (fileops.c:744)
==181176==    by 0x48FA6D4: _IO_new_file_xsputn (fileops.c:1243)
==181176==    by 0x48FA6D4: _IO_file_xsputn@@GLIBC_2.2.5 (fileops.c:1196)
==181176==    by 0x48E414C: outstring_func (vfprintf-internal.c:239)
==181176==    by 0x48E414C: __vfprintf_internal (vfprintf-internal.c:1263)
==181176==    by 0x48CF79E: printf (printf.c:33)
==181176==    by 0x10923C: program4 (program4.c:12)
==181176==    by 0x109251: main (program4.c:16)
==181176==
String: Hello World!
==181176==
```

```
==181176==
String: Hello World!
==181176==
==181176== HEAP SUMMARY:
==181176==     in use at exit: 0 bytes in 0 blocks
==181176==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
==181176==
==181176== All heap blocks were freed -- no leaks are possible
==181176==
==181176== Use --track-origins=yes to see where uninitialised values come from
==181176== For lists of detected and suppressed errors, rerun with: -s
==181176== ERROR SUMMARY: 26 errors from 4 contexts (suppressed: 0 from 0)
mohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-system-development/lab3$
```

5. Analyze Valgrind Output `program4.c`

1. **Use of Uninitialized Variable**
    - The program uses an uninitialized value in a conditional jump.

    CWE-457: Use of Uninitialized Variable – MITRE Reference

2. **Uninitialized Memory Access**
    - The program reads uninitialized memory before writing output.

    CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer – MITRE Reference

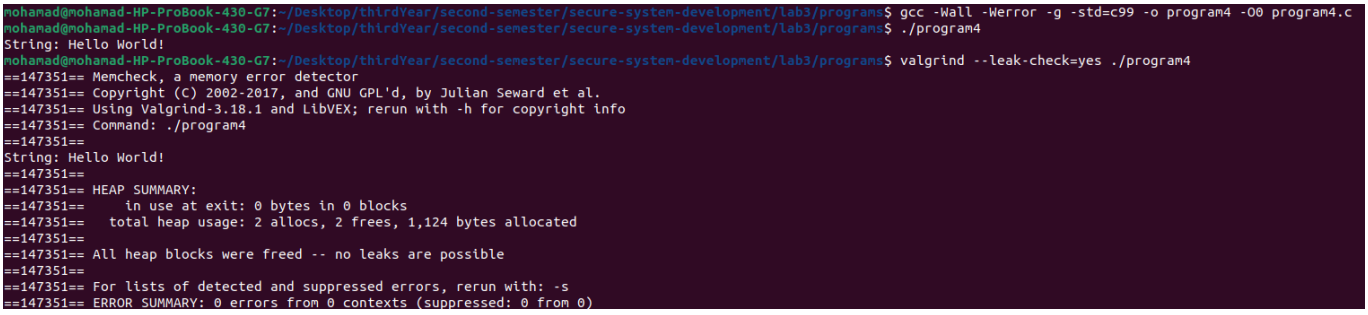
6. Propose Fixes `program4.c`

- **Fix:** The issue in the original code is returning a pointer to a local stack variable, which leads to **undefined behavior**. Instead of using a local array, allocate memory dynamically on the heap using `malloc`.
- **Safer Fix:** Use `malloc` to allocate memory and `strncpy` to safely copy the string, ensuring memory is properly managed.
- **Ensure** the allocated memory is freed after use to prevent memory leaks.

7. Verify Fixes `program4.c`

After applying the fixes, the program was recompiled and run with Valgrind:

```
gcc -Wall -Werror -g -std=c99 -o program4 -O0 program4.c
./program4
valgrind --leak-check=yes ./program4
```



Updated program4.c

Task 3 - Vulnerable HashMap Library

1. Identify and Fix CWEs

CWE-835: Loop with Unreachable Exit Condition (Infinite Loop)

- **Issue:** The `HashIndex` function had an infinite loop due to an incorrect termination condition.
- **Fix:** Corrected the loop to terminate at the end of the string.

```
for (const char* c = key; *c != '\0'; c++) {
    sum += *c;
}
```

CWE-457: Use of Uninitialized Variable

- **Issue:** `sum` in `HashIndex` was not initialized.
- **Fix:** Initialized `sum` to 0.

```
int sum = 0;
```

CWE-125: Out-of-bounds Read

- **Issue:** The hash index could exceed the array bounds.
- **Fix:** Applied modulo `MAP_MAX` to ensure valid indices.

```
return sum % MAP_MAX;
```

CWE-480: Use of Incorrect Operator

- **Issue:** `strcmp` check in `HashFind` and `HashDelete` was inverted.
- **Fix:** Corrected the condition to check for equality.

```
if (strcmp(val->KeyName, key) == 0)
```

2. Implement Improvements

- Refer to Secure Coding Guidelines for best practices.
- Updated code can be found in the repository: Updated hash.c

3. Verify Fixes

The code was compiled and run with Valgrind to ensure no memory-related issues were present.

```
 (https://dwheeler.com/secure-programs) for more information.
mohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-system-development/lab3$ flawfinder hash.c
 Flawfinder version 2.0.19, (C) 2001-2019 David A. Wheeler.
 Number of rules (primarily dangerous function names) in C/C++ ruleset: 222
 Examining hash.c

 FINAL RESULTS:


 ANALYSIS SUMMARY:

 No hits found.
 Lines analyzed = 118 in approximately 0.01 seconds (13646 lines/second)
 Physical Source Lines of Code (SLOC) = 91
 Hits@level = [0]  14 [1]   0 [2]   0 [3]   0 [4]   0 [5]   0
 Hits@level+ = [0+]  14 [1+]   0 [2+]   0 [3+]   0 [4+]   0 [5+]   0
 Hits/KSLOC@level+ = [0+] 153.846 [1+]   0 [2+]   0 [3+]   0 [4+]   0 [5+]   0
 Minimum risk level = 1

 There may be other security vulnerabilities; review your code!
 See 'Secure Programming HOWTO'
 (https://dwheeler.com/secure-programs) for more information.
mohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-system-development/lab3$
```

```
ohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-system-development/lab3/programs$ gcc -Wall -Werror -g -std=c99 -o hash -O0 hash.c
ohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-system-development/lab3/programs$ ./hash
HashInit() Successful
HashAdd(map, 'test_key')
HashAdd(map, 'test_key')
HashAdd(map, 'other_key')
HashFind(map, test_key) = {'test_key': 2}
HashDump(map) = other_key
est_key
HashDelete(map, 'test_key')
HashFind(map, test_key) = Not found
HashDump(map) = other_key
ohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-system-development/lab3/programs$ valgrind --leak-check=yes ./hash
==154075== Memcheck, a memory error detector
==154075== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==154075== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==154075== Command: ./hash
==154075==
HashInit() Successful
HashAdd(map, 'test_key')
HashAdd(map, 'test_key')
HashAdd(map, 'other_key')
HashFind(map, test_key) = {'test_key': 2}
HashDump(map) = other_key
est_key
HashDelete(map, 'test_key')
HashFind(map, test_key) = Not found
HashDump(map) = other_key
==154075==
==154075== HEAP SUMMARY:
==154075==     in use at exit: 0 bytes in 0 blocks
==154075==   total heap usage: 2 allocs, 2 frees, 2,048 bytes allocated
==154075==
==154075== All heap blocks were freed -- no leaks are possible
==154075==
==154075== For lists of detected and suppressed errors, rerun with: -s
==154075== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
ohamad@mohamad-HP-ProBook-430-G7:~/Desktop/thirdYear/second-semester/secure-system-development/lab3/programs$
```