Lab 6 - Memory Safety

CS 251, Fall 2024

Copyright Notice

Learning Goals

By the end of this lab, you'll:

- Detect and analyze memory safety errors using Address Sanitizer (ASan) for an `ourvector` class, like the one we implemented in lectures.
- Interpret Address Sanitizer error messages related to memory leaks, out-of-bounds access, null pointer dereferencing, and double-frees.
- Apply fixes to common memory management issues within the `ourvector` class.

Starter Code

lab06-starter.zip

Tasks

We strongly encourage collaboration in labs. Work with your peers, and ask your TAs questions! For this lab, complete the tasks in order.

(1) Memory Safety without ASan

Consider the following functions defined in the program in `memory.cpp`.

```
void f1() {
  int arr[5] = {1, 2, 3, 4, 5};
  for (int i = 0; i <= 5; ++i) {
    cout << arr[i] << endl;
  }
}
```

```
void f2() {
  int* ptr = nullptr;
  *ptr = 42;
}

void f3() {
  int* arr = new int[5];
  delete[] arr;
  delete[] arr;
}

void f4() {
  int* p = new int;
  *p = 5;
}

void f5() {
  int* ptr;
  *ptr = 42;
}
```

Recall the five of the common types of memory safety errors that we discussed in class:

- Memory Leaks
- Out-of-bounds Access
- Null-pointer dereferencing
- Double-free
- Uninitialized / wild pointer

There are four functions implemented in this program along with `main` (`f1` to `f5`). Each of these functions has one of the above memory safety issues. Run the program using the `make run_memory` command. This compiles and runs the program without using ASan. You should compile and run the program four different times. For each run, make a call from the `main` function to only one of the functions (`f1` to `f5`).

TODO: After executing each function, along with your partner:

a) Make note of the results of the execution of the program. Does the program give you an error?
b) Discuss what you think is the memory safety issue with that function and identify why it is happening.
c) Discuss what is problematic about that type of memory issue.

(2) Memory Leak Detection with ASan

For this exercise and all the following exercises, we have provided for you an implementation of the `ourvector` class (in `ourvector.cpp`). Now, we will be compiling using AddressSanitizer to be able to detect any memory leaks in our program.

Compile and run your program by using the `make run_ourvector` command.

TODO: Along with your partner:

a) Analyze the AddressSanitizer output by the program. There are memory leak errors in our program.
   i) Where are the leaks coming from?
   ii) How many total objects leaked? Explain why we have that number of leaked objects?
b) Use the `delete[]` command to deallocate the data that is being leaked in the appropriate place in the program.

(3) Out-of-bounds Access Detection with ASan

In your `main` function, add the following code at the bottom, so that we can compute the sum of the elements in the `ourvector` that has there been declared:

```
cout << "summation = " << sumvec(vec) << endl;
```

Here we are trying to add up the values of vec by calling the `sumvec` function which has been already implemented for you.

Run the program using the `make run_ourvector` command.

ASan is informing us that we are having a heap-buffer-overflow error, otherwise known as an out-of-bounds access. You can read the trace at the top (see image below), to identify where in your program this unsafe access is happening.

```
==76960==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60400000047c
32f68
READ of size 4 at 0x60400000047c thread T0
    #0 0x1043d0508 in ourvector::at(int) ourvector.cpp:46
    #1 0x1043d03b4 in sumvec(ourvector) ourvector.cpp:57
    #2 0x1043d06d0 in main ourvector.cpp:69
    #3 0x185ad7150  (<unknown module>)
    #4 0xa76fffffffffffc  (<unknown module>)
```

According to this output, there are two possible culprits for our problem. Is the array out-of-bounds access occurring because of an error in the `sumvec` function or perhaps the `at` function? In this case, we believe that there are memory safety issues in both.

TODO: Along with your partner:

a) Fix the problem in the `sumvec` function. After fixing this function, your program should print the sum of the elements in the vector (***note***: the summation will be printed but another error will be displayed which will be fixed in the next section).

b) What is the problem with the `at` function which allowed for the out-of-bounds access? Discuss with your partner how you would modify this `at` function so that we can handle these unsafe cases.

(4) Double-free Detection with ASan

After fixing the problem in the last section, now we are getting a different error called a double-free error. A double-free error occurs when a program tries to deallocate memory space, which has already been deallocated. In our case, this means that `delete[] data` has been called twice on the same array.

```
==77215==ERROR: AddressSanitizer: attempting double-free on 0x604000000450 in thread T0:
    #0 0x104d3d548 in _ZdaPv+0x6c (libclang_rt.asan_osx_dynamic.dylib:arm64+0x61548)
    #1 0x1044dd1ac in ourvector::~ourvector() ourvector.cpp:50
    #2 0x1044dcc88 in ourvector::~ourvector() ourvector.cpp:49
    #3 0x1044dc710 in main ourvector.cpp:70
    #4 0x185ad7150  (<unknown module>)
    #5 0xf197fffffffffc  (<unknown module>)

0x604000000450 is located 0 bytes inside of 44-byte region [0x604000000450,0x60400000047c)
freed by thread T0 here:
    #0 0x104d3d548 in _ZdaPv+0x6c (libclang_rt.asan_osx_dynamic.dylib:arm64+0x61548)
    #1 0x1044dd1ac in ourvector::~ourvector() ourvector.cpp:50
    #2 0x1044dcc88 in ourvector::~ourvector() ourvector.cpp:49
    #3 0x1044dc708 in main ourvector.cpp:69
    #4 0x185ad7150  (<unknown module>)
    #5 0xf197fffffffffc  (<unknown module>)

previously allocated by thread T0 here:
    #0 0x104d3d158 in _Znam+0x6c (libclang_rt.asan_osx_dynamic.dylib:arm64+0x61158)
    #1 0x1044dcec8 in ourvector::resize(int) ourvector.cpp:13
    #2 0x1044dc9d4 in ourvector::push_back(int) ourvector.cpp:34
    #3 0x1044dc684 in main ourvector.cpp:66
    #4 0x185ad7150  (<unknown module>)
    #5 0xf197fffffffffc  (<unknown module>)

SUMMARY: AddressSanitizer: double-free ourvector.cpp:50 in ourvector::~ourvector()
```

This output gives us valuable information that we can use to understand the error. The top trace (with the **red** heading) is giving us information about the second time that `delete[] data` was called. The second trace (with the **green** heading) is giving us information about the first time that `delete[] data` was called on the data that was double-freed.

Perhaps the line numbers for you are different than this output, but there are various important things to note here:

- They both ended up deleting `data` in the same function: `~ourvector()`, which is the destructor for `ourvector`.
- However, they originate in different places. The first `delete[]` originated in line 69 (the call to `sumvec`, and according to the image above, the other `delete[]` in line 70 (the end of `main`).

This means that `sumvec` is deallocating an array and at the end of `main`, the same array is being deallocated. The problem we are having is due to our class missing a copy constructor. Since a copy constructor is not present, then when we call `sumvec`, then `vec` is passed-by-value and the pointer to the array is copied into the parameter `v`. Since in principle, allocated data inside of a function is always deallocated once the function ends, then the same array is being deallocated in both `sumvec` and in `main` because both the parameter `v` in `sumvec` and `vec` in `main` are pointing to the same array in memory.

TODO: Along with your partner:

a) Fix this issue by implementing a copy constructor for the `ourvector` class. Make sure to compile and run your program and check if the issue has been resolved after implementation of this copy constructor. The program should print `summation = 55`.

b) In what way could we have changed the signature of the `sumvec` function, in order to avoid this double-free error altogether, even without implementing the copy constructor?

Deliverables

To record your attendance at the lab, show a TA your group's answers to the questions from each section and your the `ourvector` class with all the issues resolved. You will be asked questions about your work!

TAs will discuss solutions in the last 15 minutes of the lab. If you aren't interested in solutions, feel free to leave early once you've been checked off.

Acknowledgements

By Daniel Ayala