

debugging/valgrind/example_out_of_bound_access

Valgrind

Example: Valgrind - Out of bound access

- Source code out_of_bound_access.c++

```
#include <iostream>

int main(int argc, char *argv[]) {
    auto charArray = new char[10];
    charArray[10] = 10;
    delete [] charArray;
}
```

- Set up valgrind and build environment

```
module add \
    compiler/gnu \
    devel/valgrind
```

- Build application

```
c++ -g -ggdb out_of_bound_access.c++ -o out_of_bound_access
```

- Examine out_of_bound_access program with valgrind memchecker

```
valgrind --leak-check=full --track-origins=yes --show-reachable=yes ./out_of_bound_access

==17616== Memcheck, a memory error detector
==17616== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==17616== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==17616== Command: ./out_of_bound_access
==17616==
==17616== Invalid write of size 1
==17616==    at 0x10916E: main (out_of_bound_access.c++:5)
==17616==   Address 0x4e2208a is 0 bytes after a block of size 10 alloc'd
==17616==    at 0x48432E3: operator new[](unsigned long) (vg_replace_malloc.c:652)
==17616==   by 0x109161: main (out_of_bound_access.c++:4)
```

```
==17616==
==17616==
==17616== HEAP SUMMARY:
==17616==      in use at exit: 0 bytes in 0 blocks
==17616==    total heap usage: 2 allocs, 2 frees, 73,738 bytes allocated
==17616==
==17616== All heap blocks were freed -- no leaks are possible
==17616==
==17616== For lists of detected and suppressed errors, rerun with: -s
==17616== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```