Memory debugging

Dr. Yihsiang Liow (October 11, 2021)

Contents

1 Introduction

A memory debugging tool will help in detecting wrong use of memory. This includes using a memory before initialization, accessing an array but with index out of bound, forgetting to deallocate pointers (memory leak), etc.

There are many memory debugging tools. I'll talk about the address sanitizer (ASan) and valgrind.

```
File:  asan.tex
```

2 Address sanitizer (ASan)

The following are errors ASan can detect:

- Accessing `x[i]` where `x` is a static array or global array or pointer to an array in the heap but `i` is out of bound
- Accessing `x[i]` where `x` is a pointer that has not been allocated.
- Memory leaks (forgetting to deallocate pointer).
- Double free (deallocating deallocated pointer).
- Size mismatch deallocation (allocating a value and deallocating an array or allocating an array and deallocating a value).

ASan is an open source project from Google released to the public around mid 2011.

For installation, as root, do

```
dnf -y install libasan
```

Compile this

```
// Out of bound for stack buffer
#include <iostream>

int main()
{
    int x[10];
    int i = 10;
    int j = x[i];

    return 0;
}
```

using

```
g++ main.cpp -g -fsanitize=address
```

Run as usual with

```
./a.out
```

and you will get the following output:

```
... [snip] ...
==19927==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffc8ef4e6c8 at pc
0x000000400ab4 bp 0x7ffc8ef4e660 sp 0x7ffc8ef4e650
READ of size 4 at 0x7ffc8ef4e6c8 thread T0
    #0 0x400ab3 in main /mnt/hgfs/yliow/Documents/work/projects/valgrind/cpp/main.cpp:7
    #1 0x7ff75eec1fe9 in __libc_start_main (/lib64/libc.so.6+0x20fe9)
    #2 0x400939 in _start (/mnt/hgfs/yliow/Documents/work/projects/valgrind/cpp/a.out+0x
400939)

Address 0x7ffc8ef4e6c8 is located in stack of thread T0 at offset 72 in frame
    #0 0x4009f6 in main /mnt/hgfs/yliow/Documents/work/projects/valgrind/cpp/main.cpp:4

  This frame has 1 object(s):
    [32, 72) 'x' <== Memory access at offset 72 overflows this variable
... [snip] ...
```

This tells you that there's potentially a problem with line 7 of `main.cpp` which is

```
    int j = x[i];
```

The error message also mentioned that the error is a `stack-buffer-overflow` which tells you that you are going outside a buffer. This buffer is in the stack (i.e., in some function block and not in the heap). Furthermore the buffer belongs to a variable named `x`.

You should also test ASan with the following examples:

```
// Out of bound for stack buffer
#include <iostream>

int main()
{
    int x[10];
    int i = -1;
    int j = x[i];

    return 0;
}
```

```
// Out of bound for global buffer
```

```
#include <iostream>

int x[10];

int main()
{
    int i = -1;
    int j = x[i];

    return 0;
}
```

```
// Out of bound for global buffer

int x[10];

#include <iostream>

int main()
{
    int i = -1;
    int j = x[i];

    return 0;
}
```

```
// Out of bound for heap buffer
#include <iostream>

int main()
{
    int * x = new int[10];
    int i = 10;
    int j = x[i];
    delete [] x;
    return 0;
}
```

```
// Out of bound for heap buffer
#include <iostream>

int main()
{
    int * x = new int[10];
    int i = -1;
    int j = x[i];
    delete [] x;
    return 0;
}
```

Run the following program which has a memory leak:

```
// Memory leak
#include <iostream>

int main()
{
    int * x = new int;
    return 0;
}
```

The output is

```
=================================================================
==9843==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 4 byte(s) in 1 object(s) allocated from:
    #0 0x7f589cc99158 in operator new(unsigned long) (/lib64/libasan.so.4+0xe0158)
    #1 0x400938 in main /home/student/yliow/Documents/work/projects/valgrind/main.cpp:6
    #2 0x7f589bf39fe9 in __libc_start_main (/lib64/libc.so.6+0x20fe9)

SUMMARY: AddressSanitizer: 4 byte(s) leaked in 1 allocation(s).
```

And for this program

```
// Memory leak
#include <iostream>

int main()
{
    int * x = new int[10];
    return 0;
}
```

the output is

```
=================================================================
==9954==ERROR: LeakSanitizer: detected memory leaks

```

```
Direct leak of 40 byte(s) in 1 object(s) allocated from:
    #0 0x7fc5d4e94318 in operator new[](unsigned long) (/lib64/libasan.so.4+0xe0318)
    #1 0x400938 in main /home/student/yliow/Documents/work/projects/valgrind/main.cpp:6
    #2 0x7fc5d4134fe9 in __libc_start_main (/lib64/libc.so.6+0x20fe9)

SUMMARY: AddressSanitizer: 40 byte(s) leaked in 1 allocation(s).
```

When you run this program with a double free error:

```
// Double free - single value
#include <iostream>

int main()
{
    int * x = new int;
    delete x;
    delete x;
    return 0;
}
```

you get this output

```
=================================================================
==10033==ERROR: AddressSanitizer: attempting double-free on 0x602000000010 in thread T0:
    #0 0x7f1eb92826d8 in operator delete(void*, unsigned long) (/lib64/libasan.so.4+0xe16d8)
    #1 0x40099e in main /home/student/yliow/Documents/work/projects/valgrind/main.cpp:8
    #2 0x7f1eb8521fe9 in __libc_start_main (/lib64/libc.so.6+0x20fe9)
    #3 0x4008b9 in _start (/mnt/hgfs/yliow/Documents/work/projects/valgrind/a.out+0x4008b9)

0x602000000010 is located 0 bytes inside of 4-byte region [0x602000000010,0x602000000014)
freed by thread T0 here:
    #0 0x7f1eb92826d8 in operator delete(void*, unsigned long) (/lib64/libasan.so.4+0xe16d8)
    #1 0x40098d in main /home/student/yliow/Documents/work/projects/valgrind/main.cpp:7
    #2 0x7f1eb8521fe9 in __libc_start_main (/lib64/libc.so.6+0x20fe9)

previously allocated by thread T0 here:
    #0 0x7f1eb9281158 in operator new(unsigned long) (/lib64/libasan.so.4+0xe0158)
    #1 0x400978 in main /home/student/yliow/Documents/work/projects/valgrind/main.cpp:6
    #2 0x7f1eb8521fe9 in __libc_start_main (/lib64/libc.so.6+0x20fe9)

SUMMARY: AddressSanitizer: double-free (/lib64/libasan.so.4+0xe16d8) in operator delete(void*,
unsigned long)
==10033==ABORTING
```

You'll get a similar output for this program:

```
// Double free - array of values
#include <iostream>

int main()
{
    int * x = new int[10];
    delete [] x;
    delete [] x;
    return 0;
}
```

Here's an example of a deallocation with a wrong size:

```
// Size mismatch deallocation -- deallocation too small
#include <iostream>

int main()
{
    int * x = new int[10];
    delete x;
    return 0;
}
```

The output is

```
=================================================================
==10127==ERROR: AddressSanitizer: alloc-dealloc-mismatch (operator new [] vs operator delete) on
0x604000000010
    #0 0x7f64978656d8 in operator delete(void*, unsigned long) (/lib64/libasan.so.4+0xe16d8)
    #1 0x40098d in main /home/student/yliow/Documents/work/projects/valgrind/main.cpp:7
    #2 0x7f6496b04fe9 in __libc_start_main (/lib64/libc.so.6+0x20fe9)
    #3 0x4008b9 in _start (/mnt/hgfs/yliow/Documents/work/projects/valgrind/a.out+0x4008b9)

0x604000000010 is located 0 bytes inside of 40-byte region [0x604000000010,0x604000000038)
allocated by thread T0 here:
    #0 0x7f6497864318 in operator new[](unsigned long) (/lib64/libasan.so.4+0xe0318)
    #1 0x400978 in main /home/student/yliow/Documents/work/projects/valgrind/main.cpp:6
    #2 0x7f6496b04fe9 in __libc_start_main (/lib64/libc.so.6+0x20fe9)

SUMMARY: AddressSanitizer: alloc-dealloc-mismatch (/lib64/libasan.so.4+0xe16d8) in operator delete(void*,
unsigned long)
==10127==HINT: if you don't care about these errors you may set ASAN_OPTIONS=alloc_dealloc_mismatch=0
==10127==ABORTING
```

Try this program on your own:

```
// Size mismatch deallocation -- deallocation too big
#include <iostream>

int main()
{
    int * x = new int;
    delete [] x;
    return 0;
}
```

Here’s an example of dereferencing a pointer that has not been allocated memory:

```
// Use unallocated pointer
#include <iostream>

int main()
{
    int * x;
    int i = x[0];
    return 0;
}
```

Here’s the output:

```
ASAN:DEADLYSIGNAL
=================================================================
==10389==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000000 (pc 0x00000040096a bp 0x7fff6579a930
sp 0x7fff6579a920 T0)
==10389==The signal is caused by a READ memory access.
==10389==Hint: address points to the zero page.
    #0 0x400969 in main /home/student/yliow/Documents/work/projects/valgrind/main.cpp:7
    #1 0x7f8d74b97fe9 in __libc_start_main (/lib64/libc.so.6+0x20fe9)
    #2 0x400879 in _start (/mnt/hgfs/yliow/Documents/work/projects/valgrind/a.out+0x400879)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: SEGV /home/student/yliow/Documents/work/projects/valgrind/main.cpp:7 in main
==10389==ABORTING
```

3 Valgrind

Valgrind is similar to ASan.

Install it with

```
dnf -y install valgrind
```

Compile your C++ program using g++ with `-g` flag. Then run:

```
valgrind --tool=memcheck --leak-check=full --show-reachable=yes  \
--num-callers=20 --track-fds=yes ./a.out
```

If you want more information, you can do:

```
valgrind --tool=memcheck --leak-check=full --show-reachable=yes \
--num-callers=20 --track-fds=yes -v ./a.out
```

For instance compile this:

```
#include <iostream>

int main()
{
    int * p;
    p = new int[10];
    delete [] p;
    delete [] p;
    return 0;
}
```

using

```
g++ *.cpp -g -Wall
```

```
==22685== Memcheck, a memory error detector
==22685== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==22685== Using Valgrind-3.14.0-353a3587bb-20181007X and LibVEX; rerun with -h for copyright info
==22685== Command: ./a.out
==22685==
--22685-- Valgrind options:
--22685--    --tool=memcheck
--22685--    --leak-check=full
--22685--    --show-reachable=yes
--22685--    --num-callers=20
--22685--    --track-fds=yes
--22685--    -v
```

```
--22685-- Contents of /proc/version:
--22685--   Linux version 4.18.19-100.fc27.x86_64 (mockbuild@bkernel03.phx2.fedoraproject.org) (gcc version
7.3.1 20180712 (Red Hat 7.3.1-6) (GCC)) #1 SMP Wed Nov 14 22:04:34 UTC 2018
--22685--

... snip ...

==22685==
==22685== HEAP SUMMARY:
==22685==     in use at exit: 0 bytes in 0 blocks
==22685==   total heap usage: 2 allocs, 3 frees, 72,744 bytes allocated
==22685==
==22685== All heap blocks were freed -- no leaks are possible
==22685==
==22685== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
==22685==
==22685== 1 errors in context 1 of 1:
==22685== Invalid free() / delete / delete[] / realloc()
==22685==    at 0x4C2EAE6: operator delete[](void*) (vg_replace_malloc.c:641)
==22685==    by 0x4006C2: main (main.cpp:8)
==22685==  Address 0x5aebc80 is 0 bytes inside a block of size 40 free'd
==22685==    at 0x4C2EAE6: operator delete[](void*) (vg_replace_malloc.c:641)
==22685==    by 0x4006AF: main (main.cpp:7)
==22685==  Block was alloc'd at
==22685==    at 0x4C2DB17: operator new[](unsigned long) (vg_replace_malloc.c:423)
==22685==    by 0x400698: main (main.cpp:6)
==22685==
==22685== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Aside. There is a Valgrind tool called SGCheck (formerly known as Ptrcheck) that check stack array bounds overrun. I'll leave it to you to check it out yourself. ASan is probably enough.