Dynamic analysis for C++ (1)

Spot your bugs on the CI server.

scc

TEAMT5
杜浦數位安全

Persistent Cyber Threat Hunters

Outline

- The sanitizers
- Valgrind / Dr. Memory
- Dynamic observation tools
- RTTI for mocking
- Advanced: Fuzzing (AFL++)

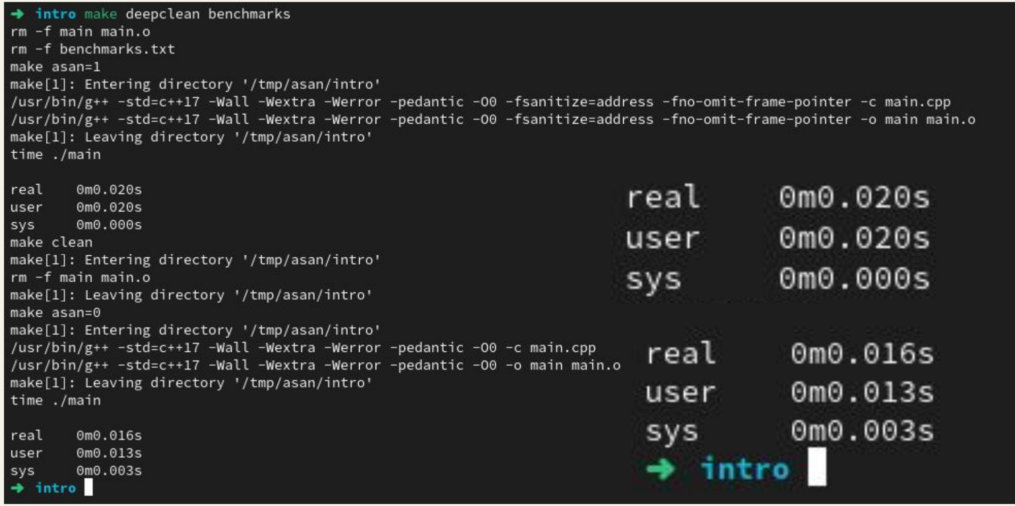
The sanitizers

- **Address sanitizer**
- **Thread sanitizer**
- **Undefined-Behavior sanitizer**

- Memory Sanitizer
- Hardware-assisted Address Sanitizer
- DataFlow Sanitizer
- …

Address sanitizer

- **About 2x slower than bare metal codes**
- Checking:
  - Out-of-bounds
  - Use-After-Free
    - stack, heap, return
  - Double Free
  - Memory Leaks

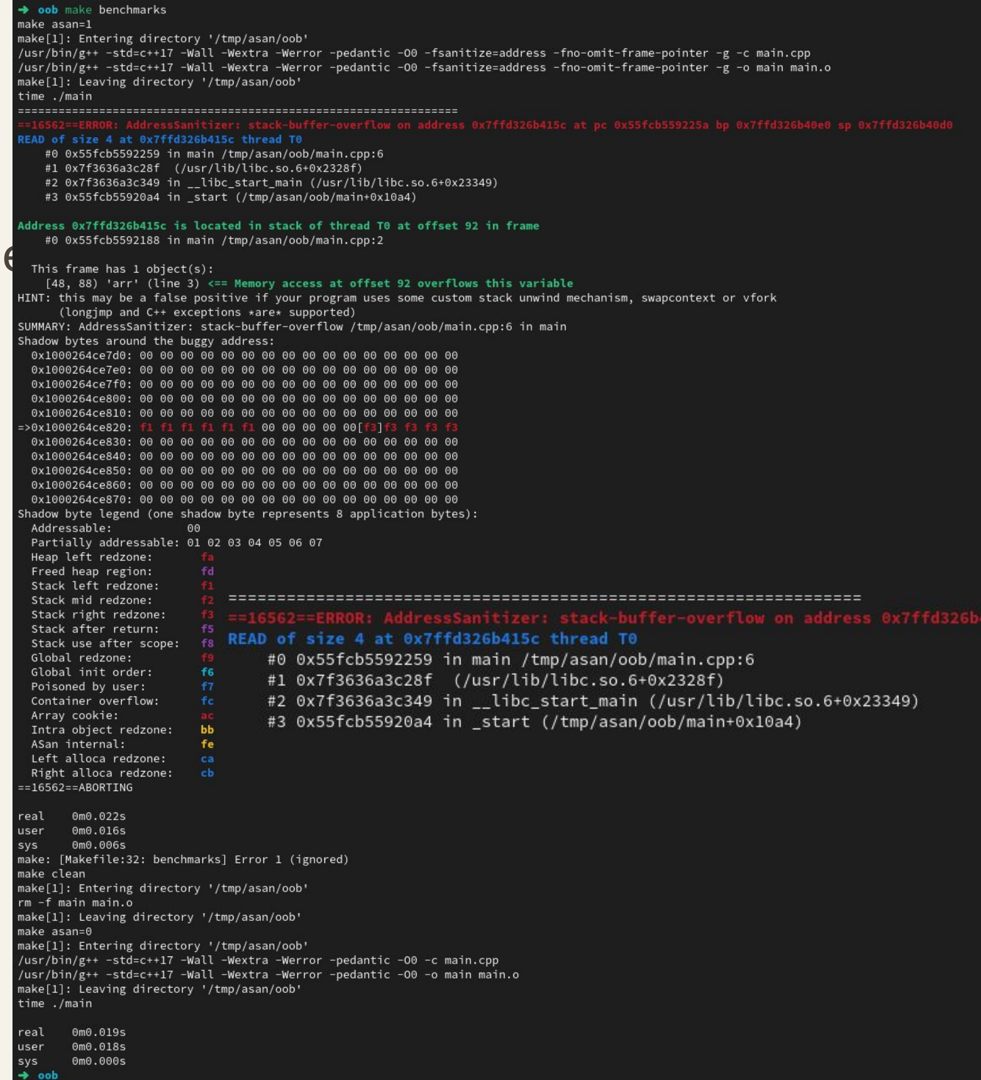


```
➜ intro make deepclean benchmarks
rm -f main main.o
rm -f benchmarks.txt
make asan=1
make[1]: Entering directory '/tmp/asan/intro'
/usr/bin/g++ -std=c++17 -Wall -Wextra -Werror -pedantic -O0 -fsanitize=address -fno-omit-frame-pointer -c main.cpp
/usr/bin/g++ -std=c++17 -Wall -Wextra -Werror -pedantic -O0 -fsanitize=address -fno-omit-frame-pointer -o main main.o
make[1]: Leaving directory '/tmp/asan/intro'
time ./main

real    0m0.020s
user    0m0.020s
sys     0m0.000s
make clean
make[1]: Entering directory '/tmp/asan/intro'
rm -f main main.o
make[1]: Leaving directory '/tmp/asan/intro'
make asan=0
make[1]: Entering directory '/tmp/asan/intro'
/usr/bin/g++ -std=c++17 -Wall -Wextra -Werror -pedantic -O0 -c main.cpp
/usr/bin/g++ -std=c++17 -Wall -Wextra -Werror -pedantic -O0 -o main main.o
make[1]: Leaving directory '/tmp/asan/intro'
time ./main

real    0m0.016s
user    0m0.013s
sys     0m0.003s
➜ intro
```

```
real    0m0.020s
user    0m0.020s
sys     0m0.000s

real    0m0.016s
user    0m0.013s
sys     0m0.003s
➜ intro
```

Address sanitizer

- About 2x slower than bare metal code
- Checking:
  - **Out-of-bounds**
  - Use-After-Free
    - stack, heap, return
  - Double Free
  - Memory Leaks
  -

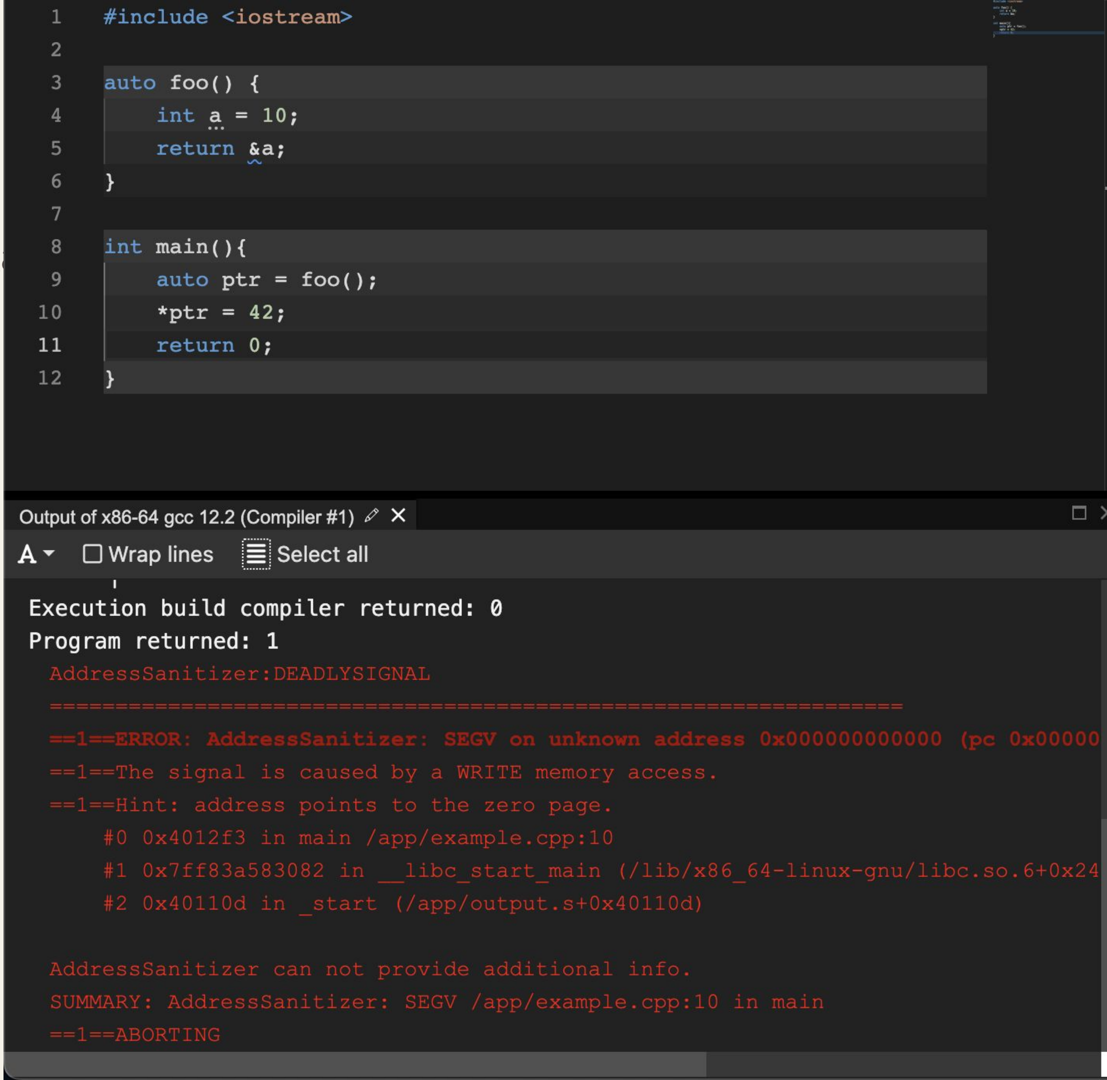


```
➜ oob make benchmarks
make asan=1
make[1]: Entering directory '/tmp/asan/oob'
/usr/bin/g++ -std=c++17 -Wall -Wextra -Werror -pedantic -O0 -fsanitize=address -fno-omit-frame-pointer -g -c main.cpp
/usr/bin/g++ -std=c++17 -Wall -Wextra -Werror -pedantic -O0 -fsanitize=address -fno-omit-frame-pointer -g -o main main.o
make[1]: Leaving directory '/tmp/asan/oob'
time ./main
=================================================================
==16562==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffd326b415c at pc 0x55fcb559225a bp 0x7ffd326b40e0 sp 0x7ffd326b40d0
READ of size 4 at 0x7ffd326b415c thread T0
    #0 0x55fcb5592259 in main /tmp/asan/oob/main.cpp:6
    #1 0x7f3636a3c28f  (/usr/lib/libc.so.6+0x2328f)
    #2 0x7f3636a3c349 in __libc_start_main (/usr/lib/libc.so.6+0x23349)
    #3 0x55fcb55920a4 in _start (/tmp/asan/oob/main+0x10a4)

Address 0x7ffd326b415c is located in stack of thread T0 at offset 92 in frame
    #0 0x55fcb5592188 in main /tmp/asan/oob/main.cpp:2

  This frame has 1 object(s):
    [48, 88) 'arr' (line 3) <== Memory access at offset 92 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism, swapcontext or vfork
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow /tmp/asan/oob/main.cpp:6 in main
Shadow bytes around the buggy address:
  0x1000264ce7d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000264ce7e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000264ce7f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000264ce800: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000264ce810: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x1000264ce820: f1 f1 f1 f1 f1 f1 00 00 00 00 00[f3]f3 f3 f3 f3
  0x1000264ce830: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000264ce840: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000264ce850: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000264ce860: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000264ce870: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
==16562==ABORTING

real    0m0.022s
user    0m0.016s
sys     0m0.006s
make: [Makefile:32: benchmarks] Error 1 (ignored)
make clean
make[1]: Entering directory '/tmp/asan/oob'
rm -f main main.o
make[1]: Leaving directory '/tmp/asan/oob'
make asan=0
make[1]: Entering directory '/tmp/asan/oob'
/usr/bin/g++ -std=c++17 -Wall -Wextra -Werror -pedantic -O0 -c main.cpp
/usr/bin/g++ -std=c++17 -Wall -Wextra -Werror -pedantic -O0 -o main main.o
make[1]: Leaving directory '/tmp/asan/oob'
time ./main

real    0m0.019s
user    0m0.018s
sys     0m0.000s
➜ oob
```

```
=================================================================
==16562==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffd326b
READ of size 4 at 0x7ffd326b415c thread T0
    #0 0x55fcb5592259 in main /tmp/asan/oob/main.cpp:6
    #1 0x7f3636a3c28f  (/usr/lib/libc.so.6+0x2328f)
    #2 0x7f3636a3c349 in __libc_start_main (/usr/lib/libc.so.6+0x23349)
    #3 0x55fcb55920a4 in _start (/tmp/asan/oob/main+0x10a4)
```

Address sanitizer

- About 2x slower than bare met
- Checking:
  - Out-of-bounds
  - Use-After-Free
    - **stack**, heap, return
  - Double Free
  - Memory Leaks
  -

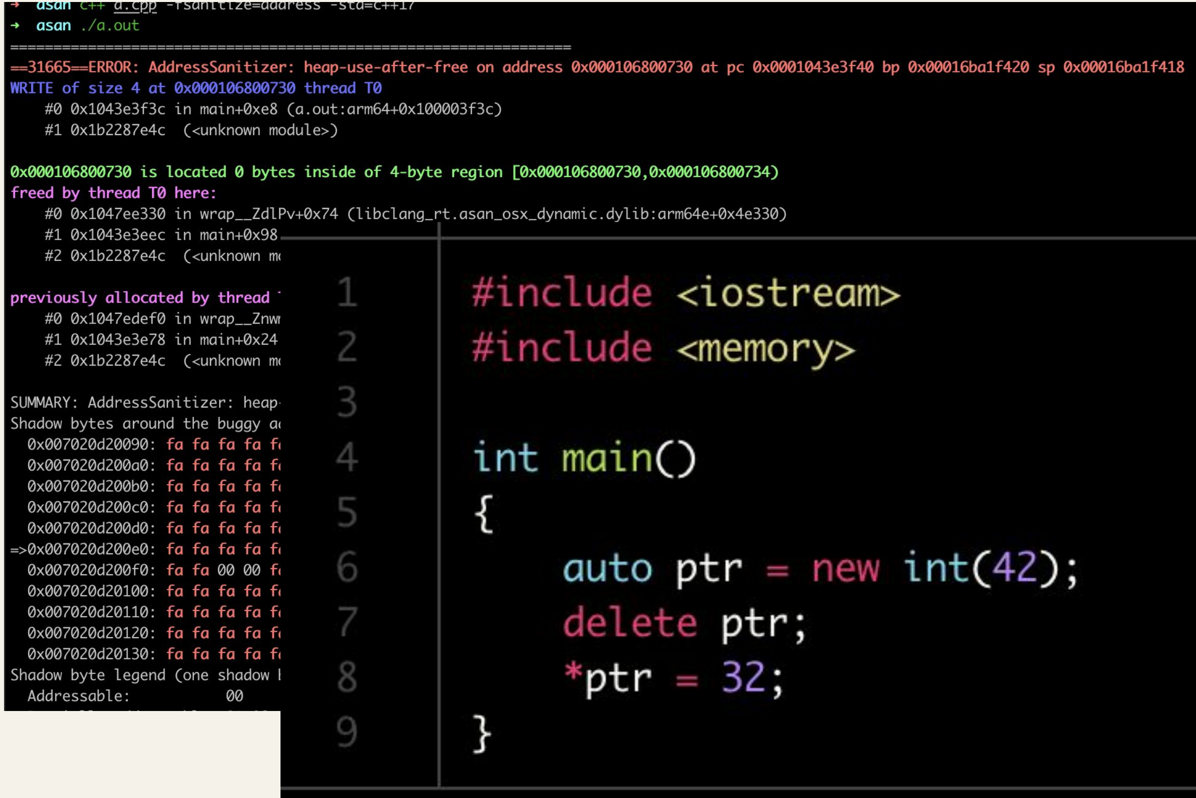


```cpp
#include <iostream>

auto foo() {
    int a = 10;
    return &a;
}

int main(){
    auto ptr = foo();
    *ptr = 42;
    return 0;
}
```

```
Execution build compiler returned: 0
Program returned: 1
  AddressSanitizer:DEADLYSIGNAL
  =================================================================
  ==1==ERROR: AddressSanitizer: SEGV on unknown address 0x000000000000 (pc 0x00000
  ==1==The signal is caused by a WRITE memory access.
  ==1==Hint: address points to the zero page.
      #0 0x4012f3 in main /app/example.cpp:10
      #1 0x7ff83a583082 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x24
      #2 0x40110d in _start (/app/output.s+0x40110d)

  AddressSanitizer can not provide additional info.
  SUMMARY: AddressSanitizer: SEGV /app/example.cpp:10 in main
  ==1==ABORTING
```

Address sanitizer

- About 2x slower than bare metal codes
- Checking:
  - Out-of-bounds
  - **Use-After-Free**
    - stack, **heap**, return
  - Double Free
  - Memory Leaks

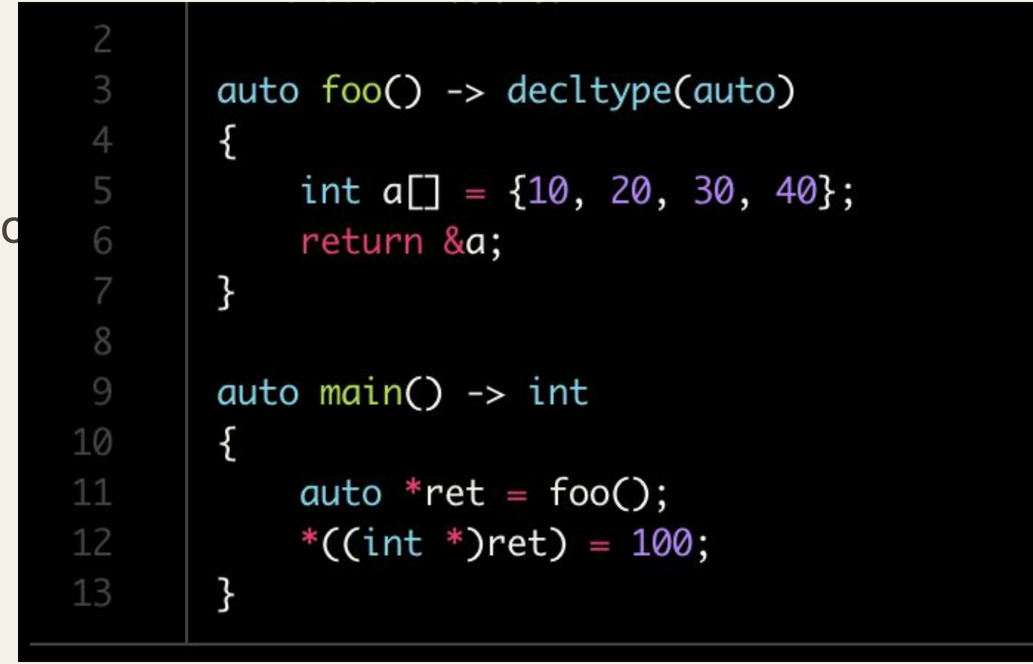


```
➜  asan c++ a.cpp -fsanitize=address -std=c++17
➜  asan ./a.out
=================================================================
==31665==ERROR: AddressSanitizer: heap-use-after-free on address 0x000106800730 at pc 0x0001043e3f40 bp 0x00016ba1f420 sp 0x00016ba1f418
WRITE of size 4 at 0x000106800730 thread T0
    #0 0x1043e3f3c in main+0xe8 (a.out:arm64+0x100003f3c)
    #1 0x1b2287e4c  (<unknown module>)

0x000106800730 is located 0 bytes inside of 4-byte region [0x000106800730,0x000106800734)
freed by thread T0 here:
    #0 0x1047ee330 in wrap__ZdlPv+0x74 (libclang_rt.asan_osx_dynamic.dylib:arm64e+0x4e330)
    #1 0x1043e3eec in main+0x98
    #2 0x1b2287e4c  (<unknown m

previously allocated by thread
    #0 0x1047edef0 in wrap__Znw
    #1 0x1043e3e78 in main+0x24
    #2 0x1b2287e4c  (<unknown m

SUMMARY: AddressSanitizer: heap
Shadow bytes around the buggy a
  0x007020d20090: fa fa fa fa f
  0x007020d200a0: fa fa fa fa f
  0x007020d200b0: fa fa fa fa f
  0x007020d200c0: fa fa fa fa f
  0x007020d200d0: fa fa fa fa f
=>0x007020d200e0: fa fa fa fa f
  0x007020d200f0: fa fa 00 00 f
  0x007020d20100: fa fa fa fa f
  0x007020d20110: fa fa fa fa f
  0x007020d20120: fa fa fa fa f
  0x007020d20130: fa fa fa fa f
Shadow byte legend (one shadow 
  Addressable:           00
```

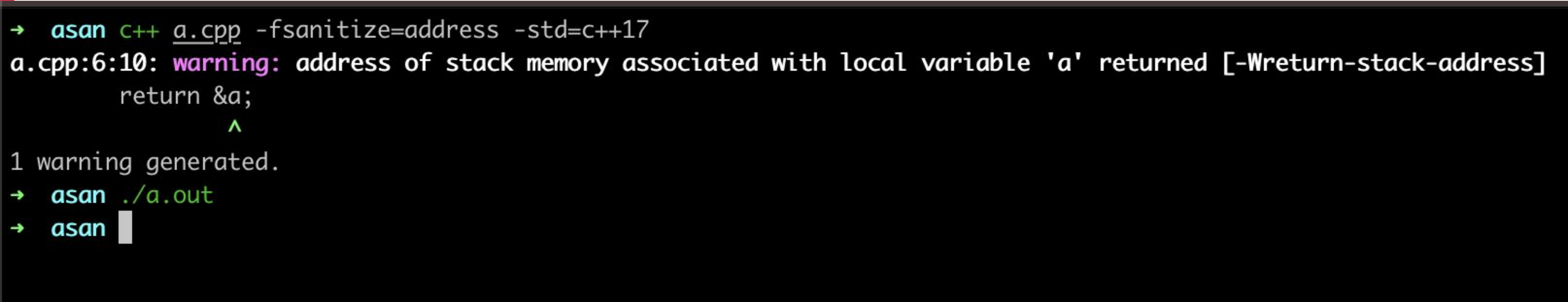
```cpp
#include <iostream>
#include <memory>

int main()
{
    auto ptr = new int(42);
    delete ptr;
    *ptr = 32;
}
```

Address sanitizer

- About 2x slower than bare metal co
- Checking:
  - Out-of-bounds
  - **Use-After-Free**
    - stack, heap, **return**
  - Double Free
  - Memory Leaks



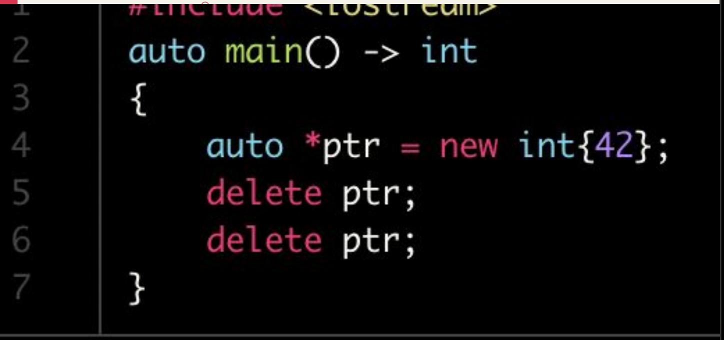
```
 2
 3  auto foo() -> decltype(auto)
 4  {
 5      int a[] = {10, 20, 30, 40};
 6      return &a;
 7  }
 8
 9  auto main() -> int
10  {
11      auto *ret = foo();
12      *((int *)ret) = 100;
13  }
```



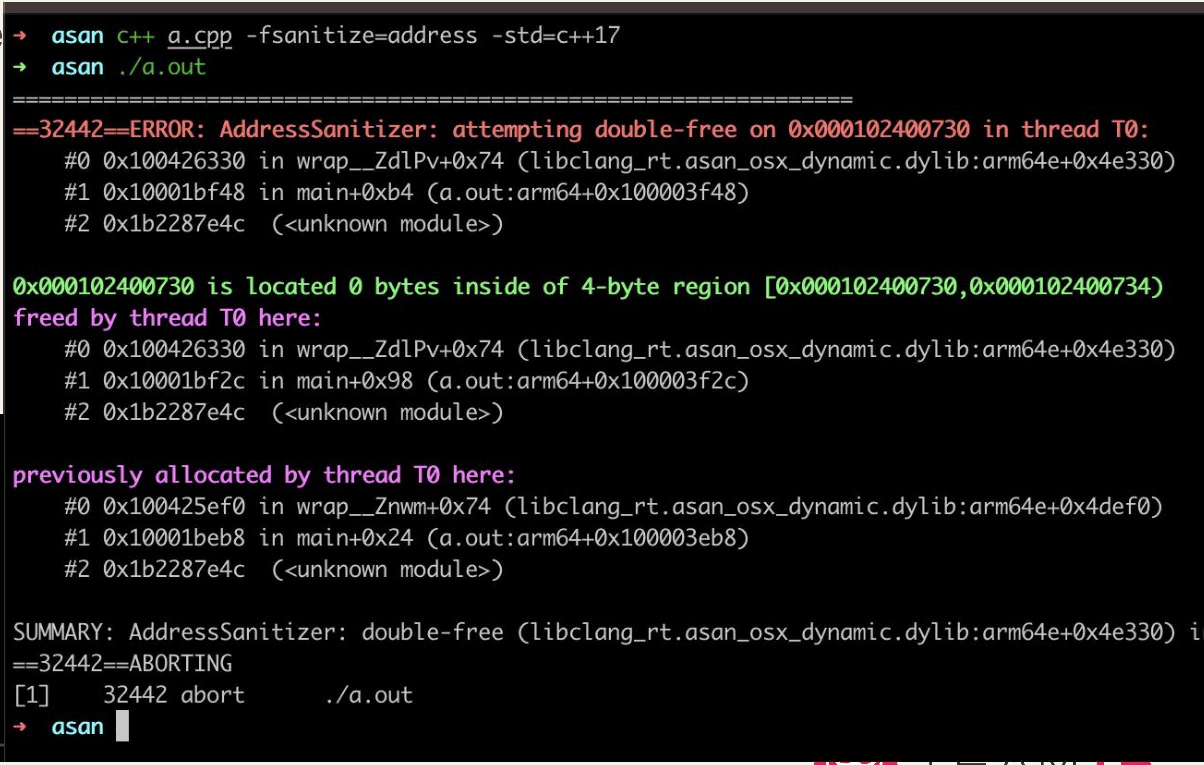
```
➜  asan c++ a.cpp -fsanitize=address -std=c++17
a.cpp:6:10: warning: address of stack memory associated with local variable 'a' returned [-Wreturn-stack-address]
        return &a;
                ^
1 warning generated.
➜  asan ./a.out
➜  asan
```

Address sanitizer

- About 2x slower than bare
- Checking:
  - Out-of-bounds
  - Use-After-Free
    - stack, heap, return
  - **Double Free**
  - Memory Leaks



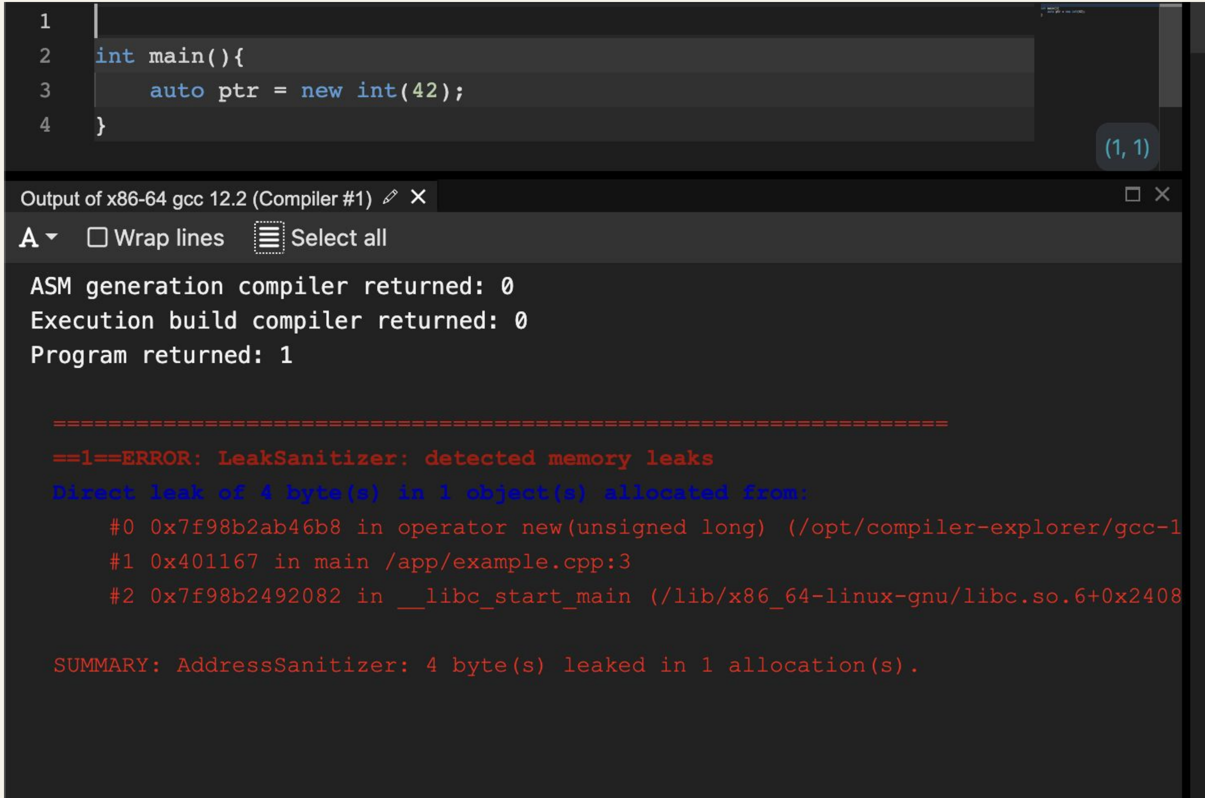
```cpp
#include <iostream>
auto main() -> int
{
    auto *ptr = new int{42};
    delete ptr;
    delete ptr;
}
```



```
➜  asan c++ a.cpp -fsanitize=address -std=c++17
➜  asan ./a.out
=================================================================
==32442==ERROR: AddressSanitizer: attempting double-free on 0x000102400730 in thread T0:
    #0 0x100426330 in wrap__ZdlPv+0x74 (libclang_rt.asan_osx_dynamic.dylib:arm64e+0x4e330)
    #1 0x10001bf48 in main+0xb4 (a.out:arm64+0x100003f48)
    #2 0x1b2287e4c  (<unknown module>)

0x000102400730 is located 0 bytes inside of 4-byte region [0x000102400730,0x000102400734)
freed by thread T0 here:
    #0 0x100426330 in wrap__ZdlPv+0x74 (libclang_rt.asan_osx_dynamic.dylib:arm64e+0x4e330)
    #1 0x10001bf2c in main+0x98 (a.out:arm64+0x100003f2c)
    #2 0x1b2287e4c  (<unknown module>)

previously allocated by thread T0 here:
    #0 0x100425ef0 in wrap__Znwm+0x74 (libclang_rt.asan_osx_dynamic.dylib:arm64e+0x4def0)
    #1 0x10001beb8 in main+0x24 (a.out:arm64+0x100003eb8)
    #2 0x1b2287e4c  (<unknown module>)

SUMMARY: AddressSanitizer: double-free (libclang_rt.asan_osx_dynamic.dylib:arm64e+0x4e330) i
==32442==ABORTING
[1]    32442 abort      ./a.out
➜  asan 
```

Address sanitizer

- About 2x slower than bare metal codes
- Checking:
  - Out-of-bounds
  - Use-After-Free
    - stack, heap, return
  - Double Free
  - **Memory Leaks**



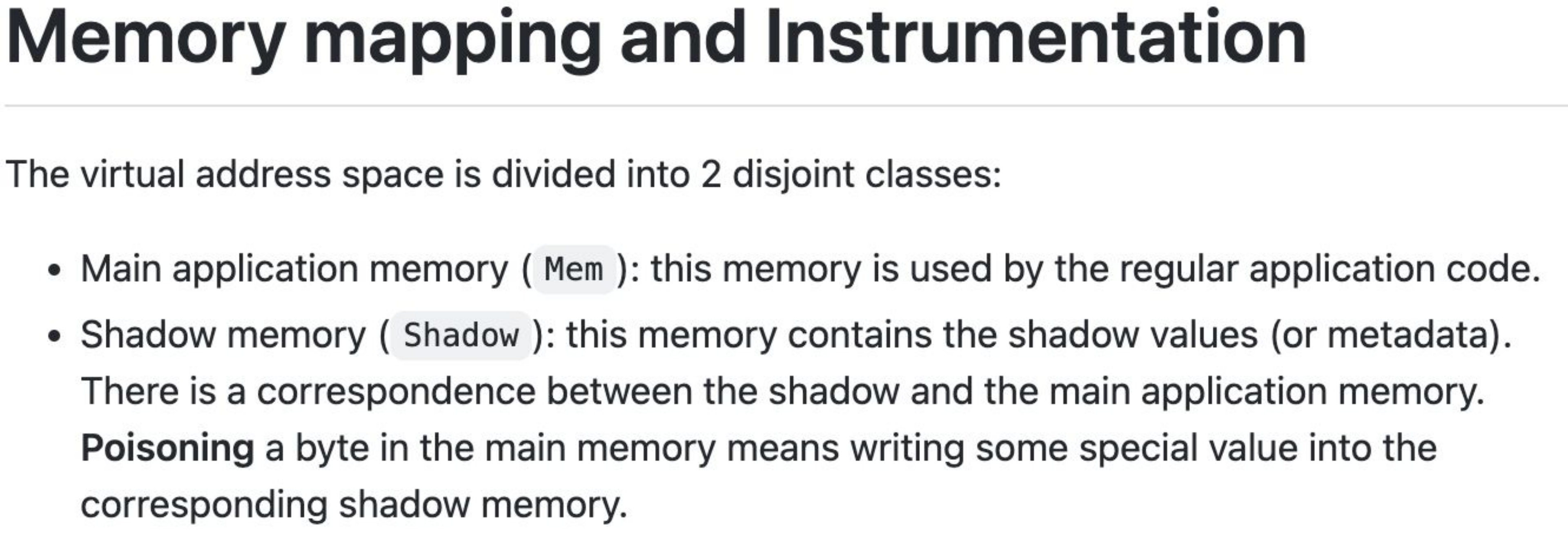
```cpp

int main(){
    auto ptr = new int(42);
}
```

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 1

  =================================================================
  ==1==ERROR: LeakSanitizer: detected memory leaks
  Direct leak of 4 byte(s) in 1 object(s) allocated from:
      #0 0x7f98b2ab46b8 in operator new(unsigned long) (/opt/compiler-explorer/gcc-1
      #1 0x401167 in main /app/example.cpp:3
      #2 0x7f98b2492082 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2408

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

Address sanitizer

- About 2x slower than bare metal codes
- Checking:
  - Out-of-bounds
  - Use-After-Free
    - stack, heap, return
  - Double Free
  - Memory Leaks
  -

Address Sanitizer Algorithm



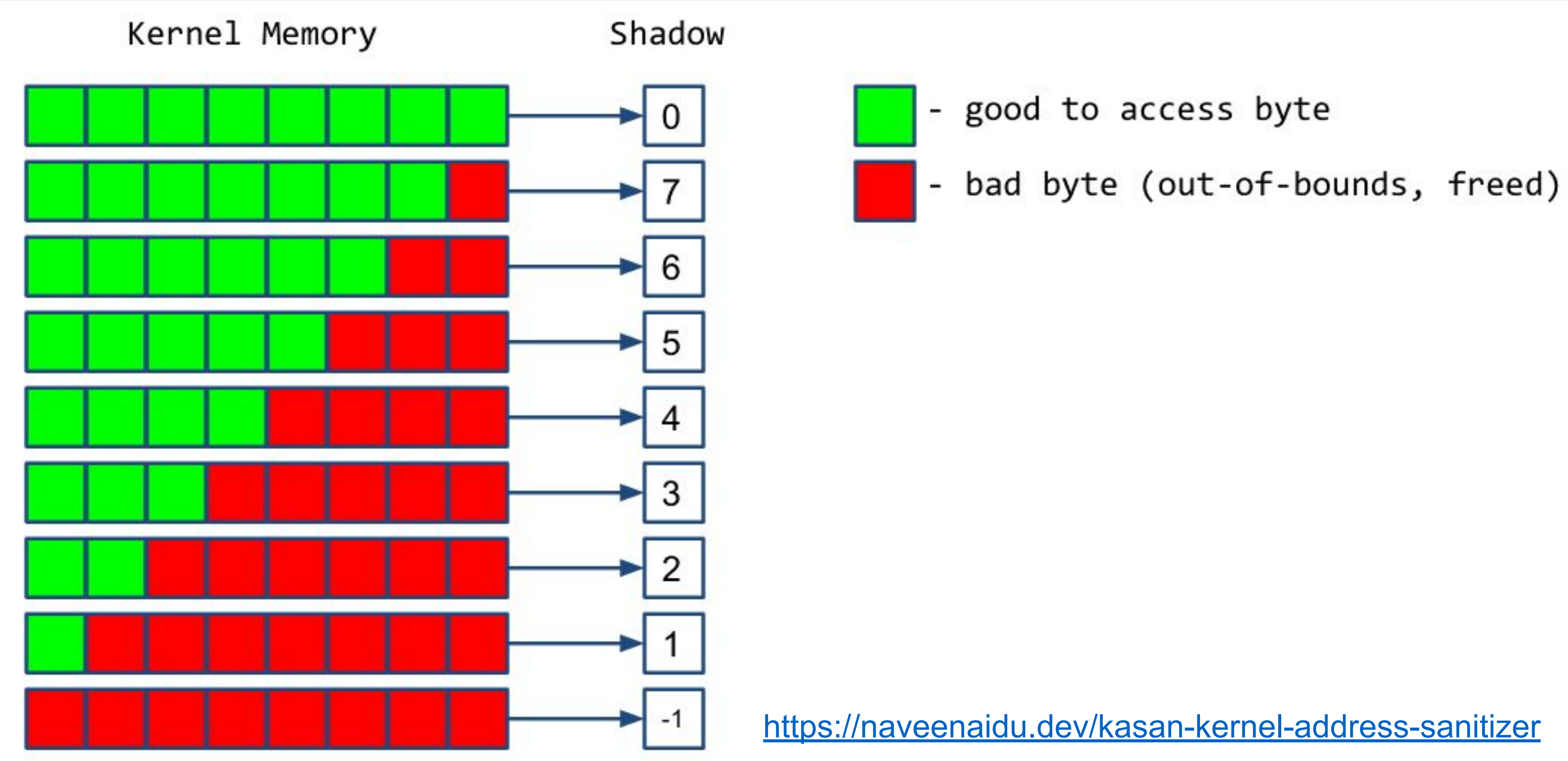
Memory mapping and Instrumentation

The virtual address space is divided into 2 disjoint classes:

- Main application memory (`Mem`): this memory is used by the regular application code.
- Shadow memory (`Shadow`): this memory contains the shadow values (or metadata). There is a correspondence between the shadow and the main application memory. **Poisoning** a byte in the main memory means writing some special value into the corresponding shadow memory.

https://github.com/google/sanitizers/wiki/AddressSanitizerAlgorithm

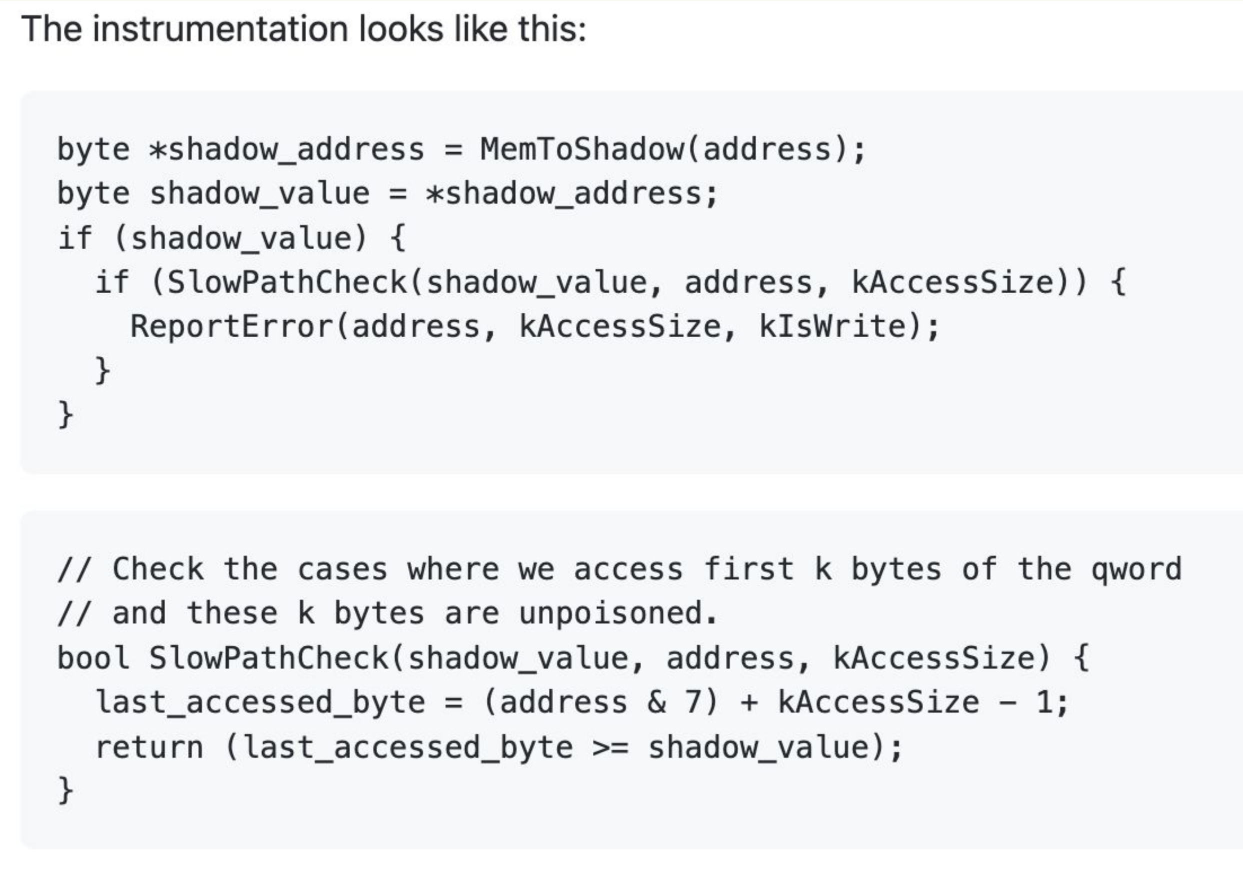
Address Sanitizer Algorithm



https://naveenaidu.dev/kasan-kernel-address-sanitizer

Address Sanitizer Algorithm

https://github.com/google/sanitizers/wiki/AddressSanitizerAlgorithm

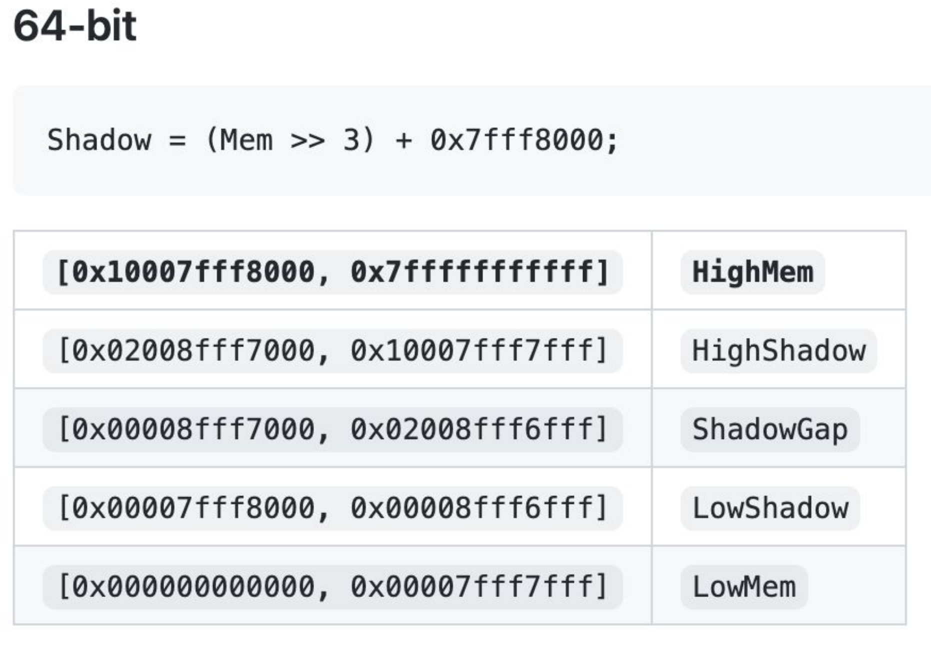


The instrumentation looks like this:

```
byte *shadow_address = MemToShadow(address);
byte shadow_value = *shadow_address;
if (shadow_value) {
  if (SlowPathCheck(shadow_value, address, kAccessSize)) {
    ReportError(address, kAccessSize, kIsWrite);
  }
}
```

```
// Check the cases where we access first k bytes of the qword
// and these k bytes are unpoisoned.
bool SlowPathCheck(shadow_value, address, kAccessSize) {
  last_accessed_byte = (address & 7) + kAccessSize - 1;
  return (last_accessed_byte >= shadow_value);
}
```

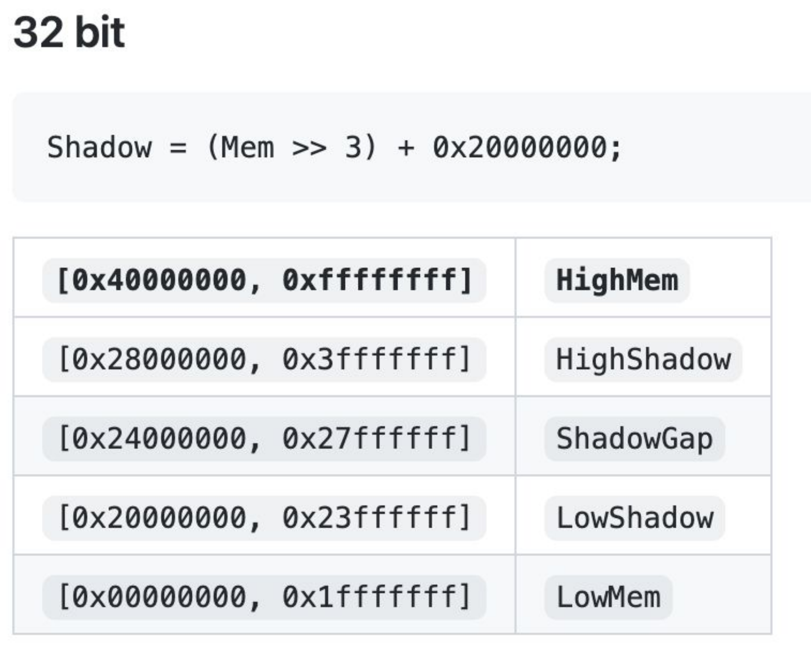
Address Sanitizer Algorithm



64-bit

```
Shadow = (Mem >> 3) + 0x7fff8000;
```

[0x10007fff8000, 0x7fffffffffff]	**HighMem**
[0x02008fff7000, 0x10007fff7fff]	HighShadow
[0x00008fff7000, 0x02008fff6fff]	ShadowGap
[0x00007fff8000, 0x00008fff6fff]	LowShadow
[0x000000000000, 0x00007fff7fff]	LowMem



32 bit

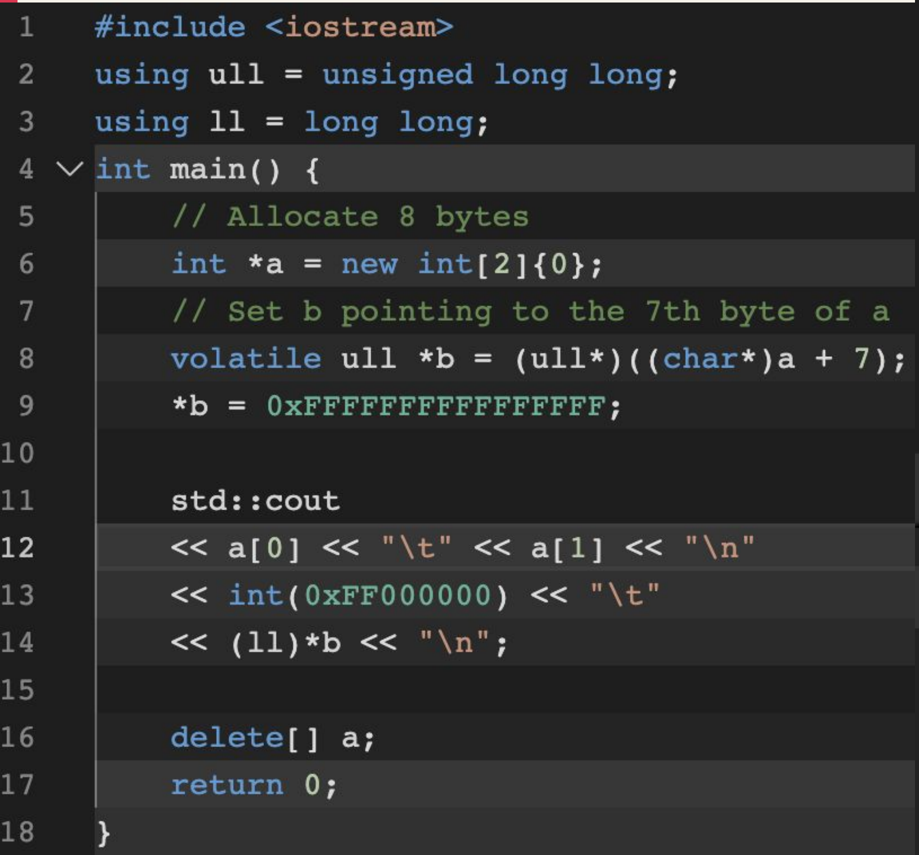
```
Shadow = (Mem >> 3) + 0x20000000;
```

[0x40000000, 0xffffffff]	**HighMem**
[0x28000000, 0x3fffffff]	HighShadow
[0x24000000, 0x27ffffff]	ShadowGap
[0x20000000, 0x23ffffff]	LowShadow
[0x00000000, 0x1fffffff]	LowMem

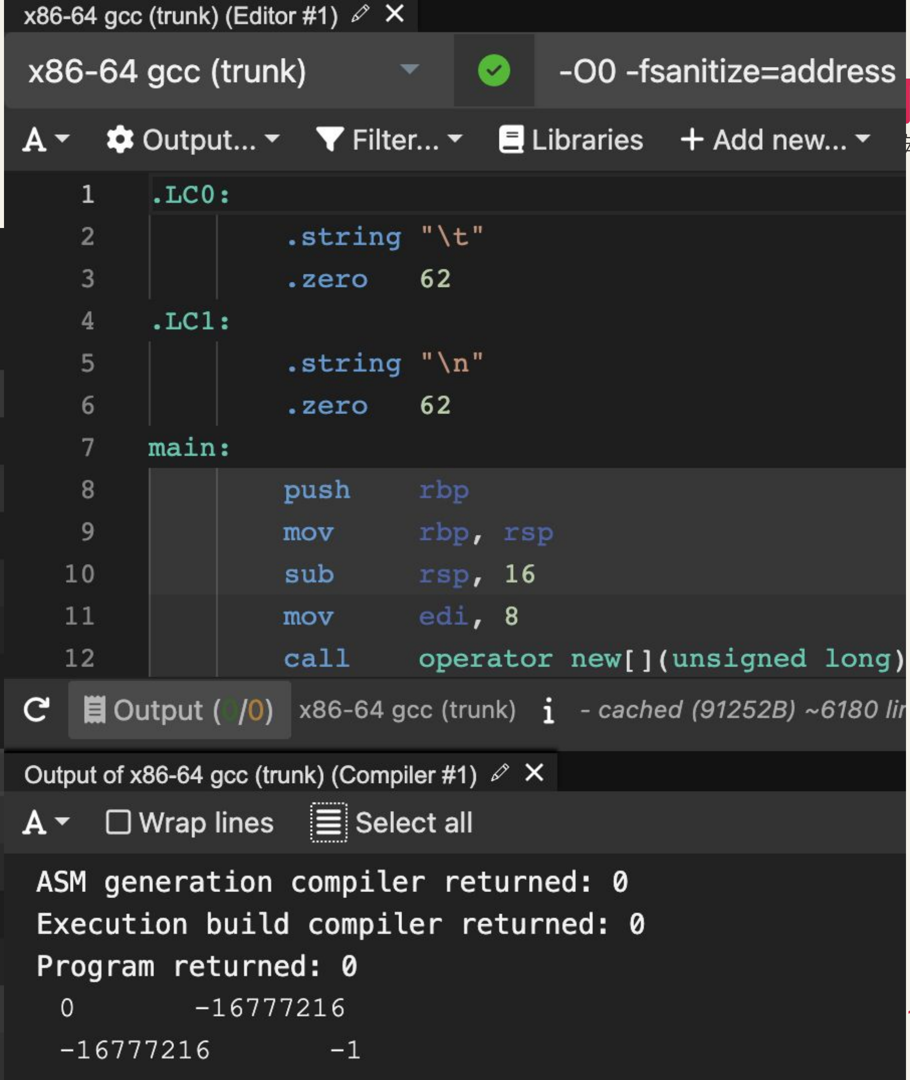
https://github.com/google/sanitizers/wiki/AddressSanitizerAlgorithm

False negative

https://godbolt.org/z/En38hne81



```cpp
#include <iostream>
using ull = unsigned long long;
using ll = long long;
int main() {
    // Allocate 8 bytes
    int *a = new int[2]{0};
    // Set b pointing to the 7th byte of a
    volatile ull *b = (ull*)((char*)a + 7);
    *b = 0xFFFFFFFFFFFFFFFF;

    std::cout
    << a[0] << "\t" << a[1] << "\n"
    << int(0xFF000000) << "\t"
    << (ll)*b << "\n";

    delete[] a;
    return 0;
}
```



-O0 -fsanitize=address

```
.LC0:
        .string "\t"
        .zero   62
.LC1:
        .string "\n"
        .zero   62
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     edi, 8
        call    operator new[](unsigned long)
```

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
 0       -16777216
 -16777216       -1
```

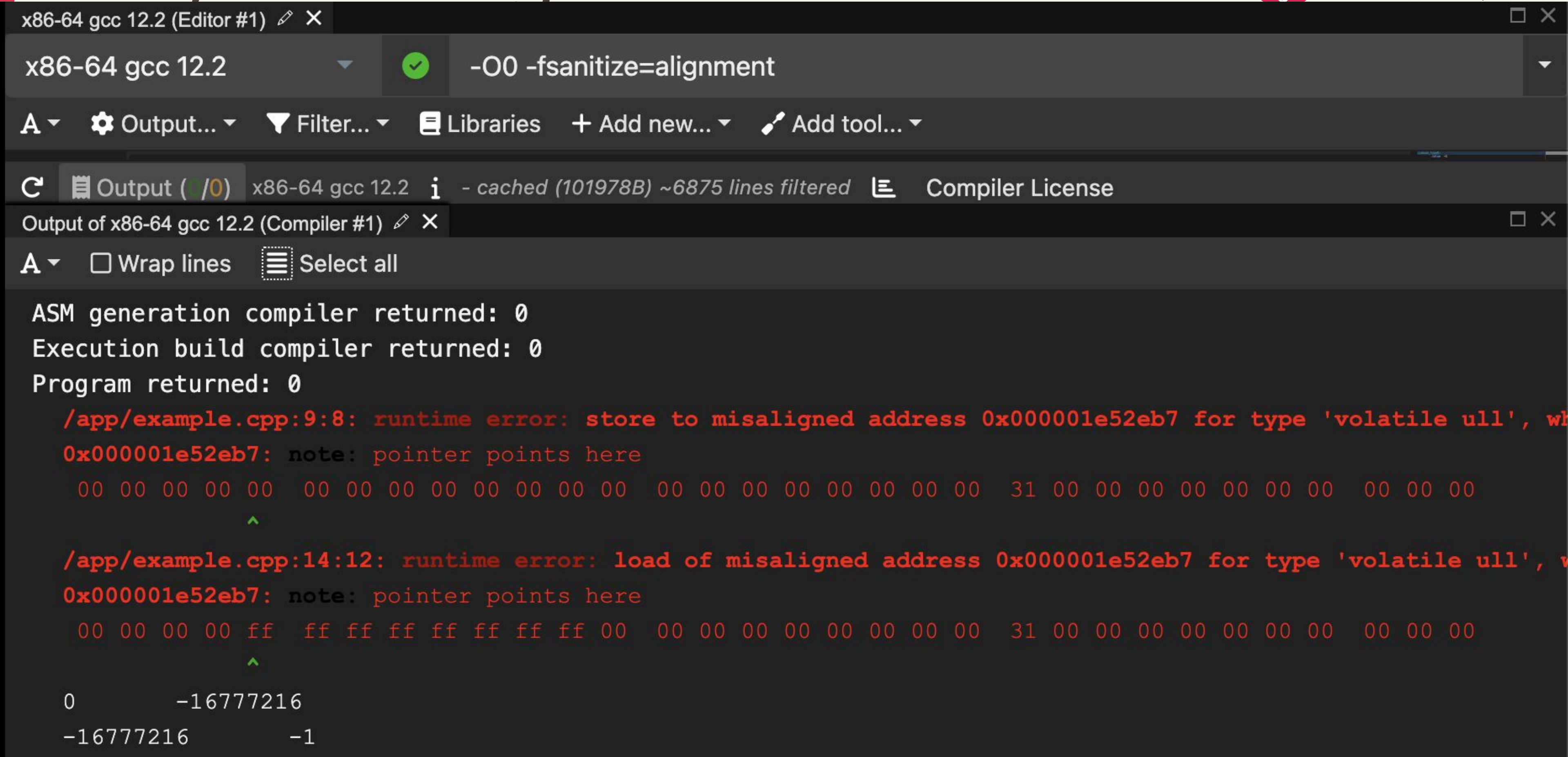
False negative

The key is:

What’s the shadow value while

DE-REFERENCING.

Only GCC alignment cloud check

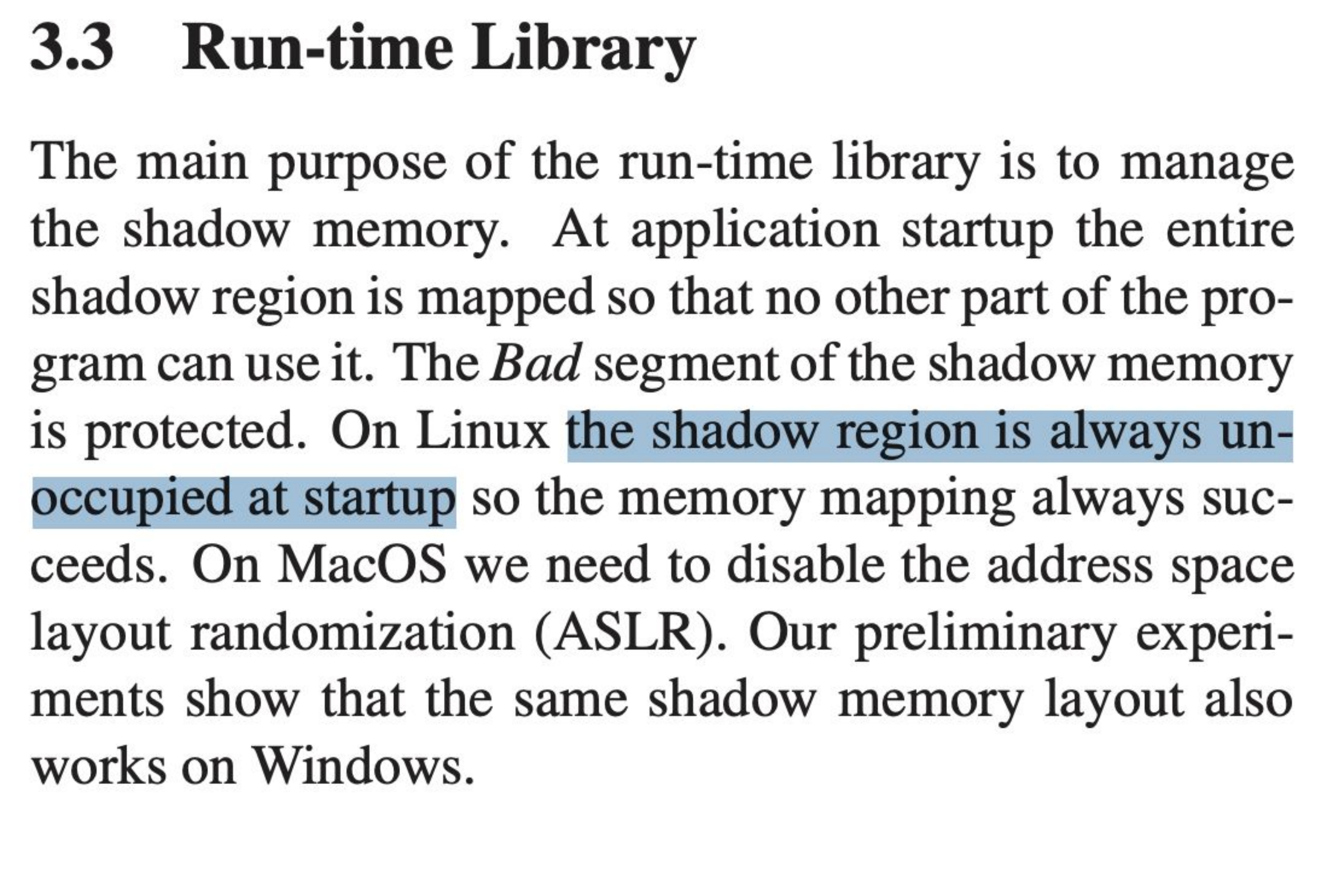


-O0 -fsanitize=alignment

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
  /app/example.cpp:9:8: runtime error: store to misaligned address 0x000001e52eb7 for type 'volatile ull', wh
  0x000001e52eb7: note: pointer points here
   00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  31 00 00 00 00 00 00 00  00 00 00
               ^
  /app/example.cpp:14:12: runtime error: load of misaligned address 0x000001e52eb7 for type 'volatile ull', w
  0x000001e52eb7: note: pointer points here
   00 00 00 00 ff  ff ff ff ff ff ff ff 00  00 00 00 00 00 00 00 00  31 00 00 00 00 00 00 00  00 00 00
               ^
  0       -16777216
  -16777216       -1
```

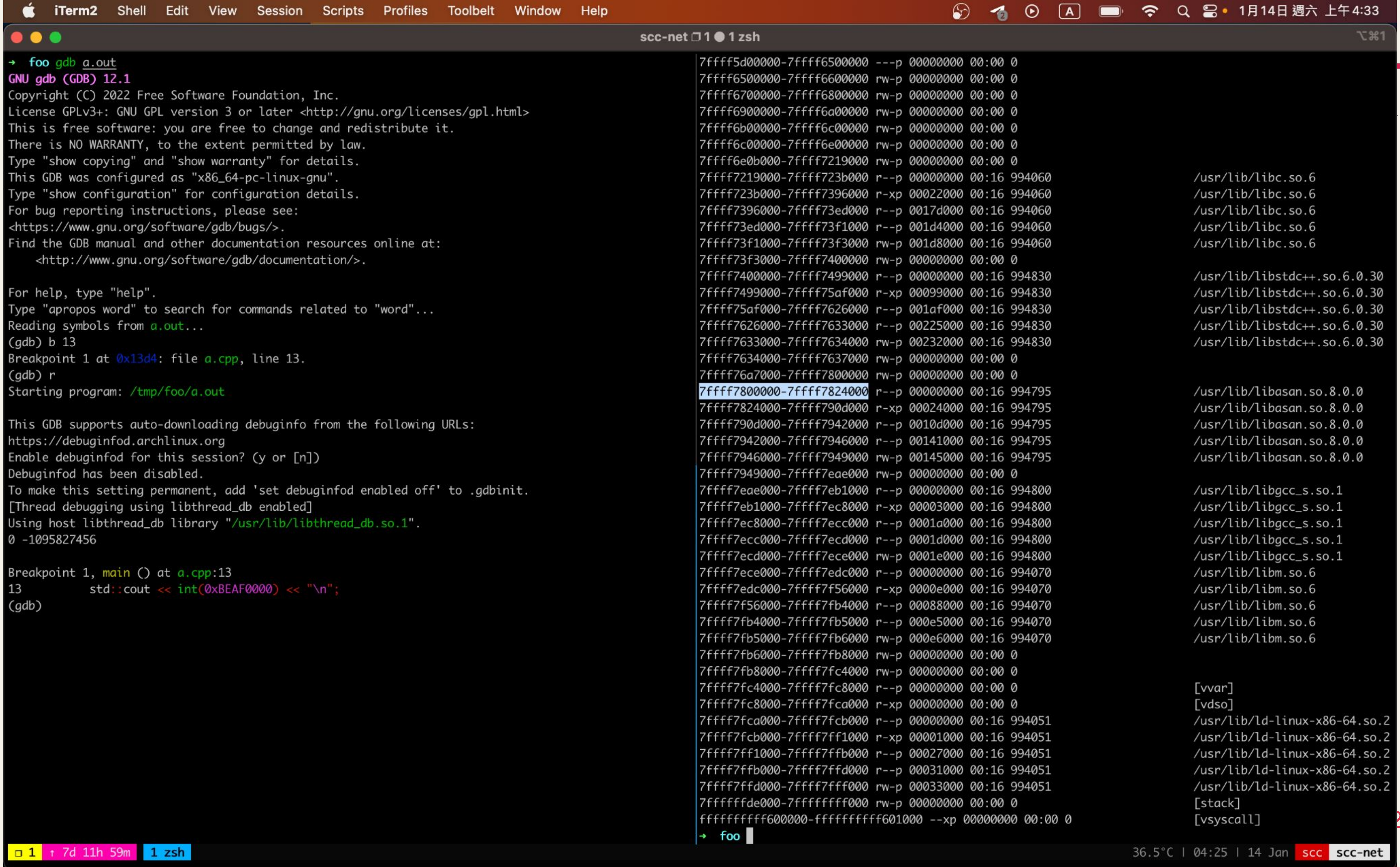
Wait, ASLR?

[1]



3.3 Run-time Library

The main purpose of the run-time library is to manage the shadow memory. At application startup the entire shadow region is mapped so that no other part of the program can use it. The *Bad* segment of the shadow memory is protected. On Linux the shadow region is always unoccupied at startup so the memory mapping always succeeds. On MacOS we need to disable the address space layout randomization (ASLR). Our preliminary experiments show that the same shadow memory layout also works on Windows.



```
→  foo gdb a.out
GNU gdb (GDB) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...
(gdb) b 13
Breakpoint 1 at 0x13d4: file a.cpp, line 13.
(gdb) r
Starting program: /tmp/foo/a.out

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.archlinux.org
Enable debuginfod for this session? (y or [n])
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/usr/lib/libthread_db.so.1".
0 -1095827456

Breakpoint 1, main () at a.cpp:13
13          std::cout << int(0xBEAF0000) << "\n";
(gdb)
```

```
7ffff5d00000-7ffff6500000 ---p 00000000 00:00 0
7ffff6500000-7ffff6600000 rw-p 00000000 00:00 0
7ffff6700000-7ffff6800000 rw-p 00000000 00:00 0
7ffff6900000-7ffff6a00000 rw-p 00000000 00:00 0
7ffff6b00000-7ffff6c00000 rw-p 00000000 00:00 0
7ffff6c00000-7ffff6e00000 rw-p 00000000 00:00 0
7ffff6e0b000-7ffff7219000 rw-p 00000000 00:00 0
7ffff7219000-7ffff723b000 r--p 00000000 00:16 994060                     /usr/lib/libc.so.6
7ffff723b000-7ffff7396000 r-xp 00022000 00:16 994060                     /usr/lib/libc.so.6
7ffff7396000-7ffff73ed000 r--p 0017d000 00:16 994060                     /usr/lib/libc.so.6
7ffff73ed000-7ffff73f1000 r--p 001d4000 00:16 994060                     /usr/lib/libc.so.6
7ffff73f1000-7ffff73f3000 rw-p 001d8000 00:16 994060                     /usr/lib/libc.so.6
7ffff73f3000-7ffff7400000 rw-p 00000000 00:00 0
7ffff7400000-7ffff7499000 r--p 00000000 00:16 994830                     /usr/lib/libstdc++.so.6.0.30
7ffff7499000-7ffff75af000 r-xp 00099000 00:16 994830                     /usr/lib/libstdc++.so.6.0.30
7ffff75af000-7ffff7626000 r--p 001af000 00:16 994830                     /usr/lib/libstdc++.so.6.0.30
7ffff7626000-7ffff7633000 r--p 00225000 00:16 994830                     /usr/lib/libstdc++.so.6.0.30
7ffff7633000-7ffff7634000 rw-p 00232000 00:16 994830                     /usr/lib/libstdc++.so.6.0.30
7ffff7634000-7ffff7637000 rw-p 00000000 00:00 0
7ffff76a7000-7ffff7800000 rw-p 00000000 00:00 0
7ffff7800000-7ffff7824000 r--p 00000000 00:16 994795                     /usr/lib/libasan.so.8.0.0
7ffff7824000-7ffff790d000 r-xp 00024000 00:16 994795                     /usr/lib/libasan.so.8.0.0
7ffff790d000-7ffff7942000 r--p 0010d000 00:16 994795                     /usr/lib/libasan.so.8.0.0
7ffff7942000-7ffff7946000 r--p 00141000 00:16 994795                     /usr/lib/libasan.so.8.0.0
7ffff7946000-7ffff7949000 rw-p 00145000 00:16 994795                     /usr/lib/libasan.so.8.0.0
7ffff7949000-7ffff7eae000 rw-p 00000000 00:00 0
7ffff7eae000-7ffff7eb1000 r--p 00000000 00:16 994800                     /usr/lib/libgcc_s.so.1
7ffff7eb1000-7ffff7ec8000 r-xp 00003000 00:16 994800                     /usr/lib/libgcc_s.so.1
7ffff7ec8000-7ffff7ecc000 r--p 0001a000 00:16 994800                     /usr/lib/libgcc_s.so.1
7ffff7ecc000-7ffff7ecd000 r--p 0001d000 00:16 994800                     /usr/lib/libgcc_s.so.1
7ffff7ecd000-7ffff7ece000 rw-p 0001e000 00:16 994800                     /usr/lib/libgcc_s.so.1
7ffff7ece000-7ffff7edc000 r--p 00000000 00:16 994070                     /usr/lib/libm.so.6
7ffff7edc000-7ffff7f56000 r-xp 0000e000 00:16 994070                     /usr/lib/libm.so.6
7ffff7f56000-7ffff7fb4000 r--p 00088000 00:16 994070                     /usr/lib/libm.so.6
7ffff7fb4000-7ffff7fb5000 r--p 000e5000 00:16 994070                     /usr/lib/libm.so.6
7ffff7fb5000-7ffff7fb6000 rw-p 000e6000 00:16 994070                     /usr/lib/libm.so.6
7ffff7fb6000-7ffff7fb8000 rw-p 00000000 00:00 0
7ffff7fb8000-7ffff7fc4000 rw-p 00000000 00:00 0
7ffff7fc4000-7ffff7fc8000 r--p 00000000 00:00 0                          [vvar]
7ffff7fc8000-7ffff7fca000 r-xp 00000000 00:00 0                          [vdso]
7ffff7fca000-7ffff7fcb000 r--p 00000000 00:16 994051                     /usr/lib/ld-linux-x86-64.so.2
7ffff7fcb000-7ffff7ff1000 r-xp 00001000 00:16 994051                     /usr/lib/ld-linux-x86-64.so.2
7ffff7ff1000-7ffff7ffb000 r--p 00027000 00:16 994051                     /usr/lib/ld-linux-x86-64.so.2
7ffff7ffb000-7ffff7ffd000 r--p 00031000 00:16 994051                     /usr/lib/ld-linux-x86-64.so.2
7ffff7ffd000-7ffff7fff000 rw-p 00033000 00:16 994051                     /usr/lib/ld-linux-x86-64.so.2
7ffffffde000-7ffffffff000 rw-p 00000000 00:00 0                          [stack]
ffffffffff600000-ffffffffff601000 --xp 00000000 00:00 0                  [vsyscall]
→  foo
```

Thread Sanitizer

Thread Sanitizer Algorithm

https://github.com/google/sanitizers/wiki/ThreadSanitizerAlgorithm

References

[1] Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitry Vyukov. 2012. AddressSanitizer: a fast address sanity checker. In Proceedings of the 2012 USENIX conference on Annual Technical Conference (USENIX ATC'12). USENIX Association, USA, 28.