Software Testing - Lab 6

網路工程研究所 310552020 陳威融

Compiler

gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)

Problem 1

Heap out-of-bounds

code

```
#include <iostream>

int main()
{
    char *array = new char[5];

    array[10] = 0;           // write out of bound
    char access = array[10]; // read out of bound

    delete[] array;

    return 0;
}
```

ASan report

```
wei@009146ab97e5:~/Software-Testing-2022/Lab_6$ g++ heapOutOfBounds.cpp -o heapOutOfBounds -fsanitize=address -g
wei@009146ab97e5:~/Software-Testing-2022/Lab_6$ ./heapOutOfBounds
=================================================================
==20998==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60200000001a at pc 0x55a6d9ddccec bp 0x7ffc96e85e30 sp 0x7ffc96e85e20
WRITE of size 1 at 0x60200000001a thread T0
    #0 0x55a6d9ddcceb in main /home/wei/Software-Testing-2022/Lab_6/heapOutOfBounds.cpp:7
    #1 0x7f8a1b593c86 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21c86)
    #2 0x55a6d9ddcbb9 in _start (/home/wei/Software-Testing-2022/Lab_6/heapOutOfBounds+0xbb9)

0x60200000001a is located 5 bytes to the right of 5-byte region [0x602000000010,0x602000000015)
allocated by thread T0 here:
    #0 0x7f8a1bdcc608 in operator new[](unsigned long) (/usr/lib/x86_64-linux-gnu/libasan.so.4+0xe0608)
    #1 0x55a6d9ddccab in main /home/wei/Software-Testing-2022/Lab_6/heapOutOfBounds.cpp:5
    #2 0x7f8a1b593c86 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21c86)

SUMMARY: AddressSanitizer: heap-buffer-overflow /home/wei/Software-Testing-2022/Lab_6/heapOutOfBounds.cpp:7 in main
Shadow bytes around the buggy address:
  0x0c047fff7fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c047fff8000: fa fa 05[fa]fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8010: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8020: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
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
==20998==ABORTING
```

valgrind report

```
wei@009146ab97e5:~/Software-Testing-2022/Lab_6$ g++ heapOutOfBounds.cpp -o heapOutOfBounds -g
wei@009146ab97e5:~/Software-Testing-2022/Lab_6$ valgrind ./heapOutOfBounds
==21373== Memcheck, a memory error detector
==21373== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==21373== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==21373== Command: ./heapOutOfBounds
==21373==
==21373== Invalid write of size 1
==21373==    at 0x1087E8: main (heapOutOfBounds.cpp:7)
==21373==  Address 0x5b7fc8a is 5 bytes after a block of size 5 alloc'd
==21373==    at 0x4C3289F: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==21373==    by 0x1087DB: main (heapOutOfBounds.cpp:5)
==21373==
==21373== Invalid read of size 1
==21373==    at 0x1087EF: main (heapOutOfBounds.cpp:8)
==21373==  Address 0x5b7fc8a is 5 bytes after a block of size 5 alloc'd
==21373==    at 0x4C3289F: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==21373==    by 0x1087DB: main (heapOutOfBounds.cpp:5)
==21373==
==21373==
==21373== HEAP SUMMARY:
==21373==     in use at exit: 0 bytes in 0 blocks
==21373==   total heap usage: 2 allocs, 2 frees, 72,709 bytes allocated
==21373==
==21373== All heap blocks were freed -- no leaks are possible
==21373==
==21373== For counts of detected and suppressed errors, rerun with: -v
==21373== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Result

ASan **能**檢測出異常 valgrind **能**檢測出異常

Stack out-of-bounds

code

```
#include <iostream>

int main()
{
    char array[5] = {0};

    array[10] = 0;           // write out of bound
    char access = array[10]; // read out of bound

    return 0;
}
```

ASan report

```
wei@009146ab97e5:~/Software-Testing-2022/Lab_6$ g++ stackOutOfBounds.cpp -o stackOutOfBounds -fsanitize=address -g -fno-stack-protector ; ./stackOutOfBounds
=================================================================
==24936==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7ffc1b20c6ba at pc 0x55a43c276df7 bp 0x7ffc1b20c670 sp 0x7ffc1b20c660
WRITE of size 1 at 0x7ffc1b20c6ba thread T0
    #0 0x55a43c276df6 in main /home/wei/Software-Testing-2022/Lab_6/stackOutOfBounds.cpp:7
    #1 0x7fb79a9f6c86 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21c86)
    #2 0x55a43c276be9 in _start (/home/wei/Software-Testing-2022/Lab_6/stackOutOfBounds+0xbe9)

Address 0x7ffc1b20c6ba is located in stack of thread T0 at offset 42 in frame
    #0 0x55a43c276cd9 in main /home/wei/Software-Testing-2022/Lab_6/stackOutOfBounds.cpp:4

  This frame has 1 object(s):
    [32, 37) 'array' <== Memory access at offset 42 overflows this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism or swapcontext
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow /home/wei/Software-Testing-2022/Lab_6/stackOutOfBounds.cpp:7 in main
Shadow bytes around the buggy address:
  0x100003639880: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100003639890: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000036398a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000036398b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000036398c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x1000036398d0: 00 00 f1 f1 f1 f1 05[f2]f2 f2 f3 f3 f3 f3 00 00
  0x1000036398e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x1000036398f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100003639900: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100003639910: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x100003639920: 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
==24936==ABORTING
```

valgrind report

```
wei@009146ab97e5:~/Software-Testing-2022/Lab_6$ g++ stackOutOfBounds.cpp -o stackOutOfBounds -g -fno-stack-protector; valgrind ./stackOutOfBounds
==24519== Memcheck, a memory error detector
==24519== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==24519== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==24519== Command: ./stackOutOfBounds
==24519==
==24519==
==24519== HEAP SUMMARY:
==24519==     in use at exit: 0 bytes in 0 blocks
==24519==   total heap usage: 1 allocs, 1 frees, 72,704 bytes allocated
==24519==
==24519== All heap blocks were freed -- no leaks are possible
==24519==
==24519== For counts of detected and suppressed errors, rerun with: -v
==24519== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Result

ASan **能**檢測出異常 valgrind **不能**檢測出異常

Global out-of-bounds

code

```
#include <iostream>

char global_array[5] = {0};
```

```
int main()
{

    global_array[10] = 0;            // write out of bound
    char access = global_array[10]; // read out of bound

    return 0;
}
```

ASan report

```
wei@009146ab97e5:~/Software-Testing-2022/Lab_6$ make asan
g++ -g -fno-stack-protector globalOutOfBounds.cpp -o globalOutOfBounds -fsanitize=address
./globalOutOfBounds
=================================================================
==27810==ERROR: AddressSanitizer: global-buffer-overflow on address 0x559b1a5ba18a at pc 0x559b1a3b8c89 bp 0x7ffd4ec65820 sp 0x7ffd4ec65810
WRITE of size 1 at 0x559b1a5ba18a thread T0
    #0 0x559b1a3b8c88 in main /home/wei/Software-Testing-2022/Lab_6/globalOutOfBounds.cpp:8
    #1 0x7f5ebdfecc86 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21c86)
    #2 0x559b1a3b8b69 in _start (/home/wei/Software-Testing-2022/Lab_6/globalOutOfBounds+0xb69)

0x559b1a5ba18a is located 5 bytes to the right of global variable 'global_array' defined in 'globalOutOfBounds.cpp:3:6' (0x559b1a5ba180) of size 5
SUMMARY: AddressSanitizer: global-buffer-overflow /home/wei/Software-Testing-2022/Lab_6/globalOutOfBounds.cpp:8 in main
Shadow bytes around the buggy address:
  0x0ab3e34af3e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ab3e34af3f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ab3e34af400: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ab3e34af410: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ab3e34af420: 00 00 00 00 00 00 00 00 01 f9 f9 f9 f9 f9 f9 f9
=>0x0ab3e34af430: 05[f9]f9 f9 f9 f9 f9 f9 00 00 00 00 00 00 00 00
  0x0ab3e34af440: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ab3e34af450: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ab3e34af460: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ab3e34af470: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ab3e34af480: 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
==27810==ABORTING
```

valgrind report

```
wei@009146ab97e5:~/Software-Testing-2022/Lab_6$ make valgrind
g++ -g -fno-stack-protector globalOutOfBounds.cpp -o globalOutOfBounds
valgrind ./globalOutOfBounds
==28108== Memcheck, a memory error detector
==28108== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==28108== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==28108== Command: ./globalOutOfBounds
==28108==
==28108==
==28108== HEAP SUMMARY:
==28108==     in use at exit: 0 bytes in 0 blocks
==28108==   total heap usage: 1 allocs, 1 frees, 72,704 bytes allocated
==28108==
==28108== All heap blocks were freed -- no leaks are possible
==28108==
==28108== For counts of detected and suppressed errors, rerun with: -v
==28108== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Result

ASan **能**檢測出異常 valgrind **不能**檢測出異常

Use-after-free

code

```
#include <iostream>

int main()
{
    char *array = new char[5];

    delete[] array;

    array[0] = 0;           // write after free
    char access = array[0]; // read after free

    return 0;
}
```

ASan report

```
wei@009146ab97e5:~/Software-Testing-2022/Lab_6$ make asan
g++     useAfterFree.cpp   -o useAfterFree
g++ -g -fno-stack-protector useAfterFree.cpp -o useAfterFree -fsanitize=address
./useAfterFree
=================================================================
==29070==ERROR: AddressSanitizer: heap-use-after-free on address 0x602000000010 at pc 0x55e9ce789ca7 bp 0x7ffde97df3c0 sp 0x7ffde97df3b0
WRITE of size 1 at 0x602000000010 thread T0
    #0 0x55e9ce789ca6 in main /home/wei/Software-Testing-2022/Lab_6/useAfterFree.cpp:9
    #1 0x7f74a5b7bc86 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21c86)
    #2 0x55e9ce789b69 in _start (/home/wei/Software-Testing-2022/Lab_6/useAfterFree+0xb69)

0x602000000010 is located 0 bytes inside of 5-byte region [0x602000000010,0x602000000015)
freed by thread T0 here:
    #0 0x7f74a63b5480 in operator delete[](void*) (/usr/lib/x86_64-linux-gnu/libasan.so.4+0xe1480)
    #1 0x55e9ce789c72 in main /home/wei/Software-Testing-2022/Lab_6/useAfterFree.cpp:7
    #2 0x7f74a5b7bc86 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21c86)

previously allocated by thread T0 here:
    #0 0x7f74a63b4608 in operator new[](unsigned long) (/usr/lib/x86_64-linux-gnu/libasan.so.4+0xe0608)
    #1 0x55e9ce789c5b in main /home/wei/Software-Testing-2022/Lab_6/useAfterFree.cpp:5
    #2 0x7f74a5b7bc86 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21c86)

SUMMARY: AddressSanitizer: heap-use-after-free /home/wei/Software-Testing-2022/Lab_6/useAfterFree.cpp:9 in main
Shadow bytes around the buggy address:
  0x0c047fff7fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c047fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c047fff8000: fa fa[fd]fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8010: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8020: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c047fff8050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
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
==29070==ABORTING
```

valgrind report

```
wei@009146ab97e5:~/Software-Testing-2022/Lab_6$ make valgrind
g++ -g -fno-stack-protector useAfterFree.cpp -o useAfterFree
valgrind ./useAfterFree
==29322== Memcheck, a memory error detector
==29322== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==29322== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==29322== Command: ./useAfterFree
==29322==
==29322== Invalid write of size 1
==29322==    at 0x1087F7: main (useAfterFree.cpp:9)
==29322==  Address 0x5b7fc80 is 0 bytes inside a block of size 5 free'd
==29322==    at 0x4C3373B: operator delete[](void*) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==29322==    by 0x1087F2: main (useAfterFree.cpp:7)
==29322==  Block was alloc'd at
==29322==    at 0x4C3289F: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==29322==    by 0x1087DB: main (useAfterFree.cpp:5)
==29322==
==29322== Invalid read of size 1
==29322==    at 0x1087FE: main (useAfterFree.cpp:10)
==29322==  Address 0x5b7fc80 is 0 bytes inside a block of size 5 free'd
==29322==    at 0x4C3373B: operator delete[](void*) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==29322==    by 0x1087F2: main (useAfterFree.cpp:7)
==29322==  Block was alloc'd at
==29322==    at 0x4C3289F: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==29322==    by 0x1087DB: main (useAfterFree.cpp:5)
==29322==
==29322==
==29322== HEAP SUMMARY:
==29322==     in use at exit: 0 bytes in 0 blocks
==29322==   total heap usage: 2 allocs, 2 frees, 72,709 bytes allocated
==29322==
==29322== All heap blocks were freed -- no leaks are possible
==29322==
==29322== For counts of detected and suppressed errors, rerun with: -v
==29322== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Result

ASan **能**檢測出異常 valgrind **能**檢測出異常

Use-after-return

code

```
// export ASAN_OPTIONS=detect_stack_use_after_return=1

char *x;

void foo()
{
    char stack_buffer[42];
    x = &stack_buffer[13];
}

int main()
{
    foo();
    *x = 42; // Boom!
```

```
    return 0;
}
```

ASan report

```
wei@009146ab97e5:~/Software-Testing-2022/Lab_6$ make asan
g++ -g -fno-stack-protector useAfterReturn.cpp -o useAfterReturn -fsanitize=address
./useAfterReturn
=================================================================
==32505==ERROR: AddressSanitizer: stack-use-after-return on address 0x7f3b8030002d at pc 0x55b197e76b39 bp 0x7fffbbc53140 sp 0x7fffbbc53130
WRITE of size 1 at 0x7f3b8030002d thread T0
    #0 0x55b197e76b38 in main /home/wei/Software-Testing-2022/Lab_6/useAfterReturn.cpp:14
    #1 0x7f3b84427c86 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21c86)
    #2 0x55b197e76939 in _start (/home/wei/Software-Testing-2022/Lab_6/useAfterReturn+0x939)

Address 0x7f3b8030002d is located in stack of thread T0 at offset 45 in frame
    #0 0x55b197e76a29 in foo() /home/wei/Software-Testing-2022/Lab_6/useAfterReturn.cpp:6

  This frame has 1 object(s):
    [32, 74) 'stack_buffer' <== Memory access at offset 45 is inside this variable
HINT: this may be a false positive if your program uses some custom stack unwind mechanism or swapcontext
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-use-after-return /home/wei/Software-Testing-2022/Lab_6/useAfterReturn.cpp:14 in main
Shadow bytes around the buggy address:
  0x0fe7f0057fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe7f0057fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe7f0057fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe7f0057fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe7f0057ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0fe7f0058000: f5 f5 f5 f5 f5[f5]f5 f5 f5 f5 f5 f5 f5 f5 f5 f5
  0x0fe7f0058010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe7f0058020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe7f0058030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe7f0058040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe7f0058050: 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
==32505==ABORTING
```

valgrind report

```
wei@009146ab97e5:~/Software-Testing-2022/Lab_6$ make valgrind
g++ -g -fno-stack-protector useAfterReturn.cpp -o useAfterReturn
valgrind ./useAfterReturn
==32750== Memcheck, a memory error detector
==32750== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==32750== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==32750== Command: ./useAfterReturn
==32750==
==32750==
==32750== HEAP SUMMARY:
==32750==     in use at exit: 0 bytes in 0 blocks
==32750==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==32750==
==32750== All heap blocks were freed -- no leaks are possible
==32750==
==32750== For counts of detected and suppressed errors, rerun with: -v
==32750== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Result

ASan **能**檢測出異常 valgrind **不能**檢測出異常

Problem 2

寫一個簡單程式 with ASan，Stack buffer overflow 剛好越過redzone(並沒有對 redzone 做讀寫)，並說明 ASan 能否找的出來？

code

```
#include <iostream>

int main()
{
    char a[8] = {0};
    char b[8] = {0};

    a[10] = 0;           // write out of bound
    char access = a[10]; // read out of bound

    return 0;
}
```

Result

```
wei@009146ab97e5:~/Software-Testing-2022/Lab_6$ make asan
g++     crossRedZone.cpp   -o crossRedZone
g++ -g -fno-stack-protector crossRedZone.cpp -o crossRedZone -fsanitize=address
./crossRedZone
=================================================================
==3539==ERROR: AddressSanitizer: stack-buffer-overflow on address 0x7fc59b30002a at pc 0x5621220f5dcc bp 0x7fffe275f840 sp 0x7fffe275f830
WRITE of size 1 at 0x7fc59b30002a thread T0
    #0 0x5621220f5dcb in main /home/wei/Software-Testing-2022/Lab_6/crossRedZone.cpp:8
    #1 0x7fc59f340c86 in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x21c86)
    #2 0x5621220f5be9 in _start (/home/wei/Software-Testing-2022/Lab_6/crossRedZone+0xbe9)

Address 0x7fc59b30002a is located in stack of thread T0 at offset 42 in frame
    #0 0x5621220f5cd9 in main /home/wei/Software-Testing-2022/Lab_6/crossRedZone.cpp:4

  This frame has 2 object(s):
    [32, 40) 'a' <== Memory access at offset 42 overflows this variable
    [96, 104) 'b'
HINT: this may be a false positive if your program uses some custom stack unwind mechanism or swapcontext
      (longjmp and C++ exceptions *are* supported)
SUMMARY: AddressSanitizer: stack-buffer-overflow /home/wei/Software-Testing-2022/Lab_6/crossRedZone.cpp:8 in main
Shadow bytes around the buggy address:
  0x0ff933657fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff933657fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff933657fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff933657fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff933657ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0ff933658000: f1 f1 f1 f1 00[f2]f2 f2 f2 f2 f2 f2 00 f2 f2 f2
  0x0ff933658010: f3 f3 f3 f3 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff933658020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff933658030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff933658040: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0ff933658050: 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
==3539==ABORTING
```

由上述實驗可知，a的記憶體位置為[32, 40)、b的記憶體位置為[96, 104)，因此若將a的起始位置+64即可跨越readzone，實驗結果如下：

```
#include <iostream>

int main()
{
    char a[8] = {0};
    char b[8] = {0};

    a[64] = 0;            // write out of bound
    char access = a[64]; // read out of bound

    return 0;
}
```

Result

```
wei@009146ab97e5:~/Software-Testing-2022/Lab_6$ make asan
g++     crossRedZone.cpp    -o crossRedZone
g++ -g -fno-stack-protector crossRedZone.cpp -o crossRedZone -fsanitize=address
./crossRedZone
```

Conclusion

ASan無法找出剛好跨越readzone的錯誤