Software Testing Lab6

Environment

```
gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
```

AddressSanitizer (ASan)

```
$ gcc -fsanitize=address -g -o file file.c
$ ./file
```

Valgrind

```
$ gcc -o test test.c
$ valgrind ./test

```

Part1

Heap out-of-bounds read/write

Source code

```
#include <stdlib.h>
#include <stdio.h>

int main(){
    int length = 4;
    int *p = (int*) malloc(length * sizeof(int));

    p[4] = 4;
    printf("%d", p[4]);

    return 0;
}
```

ASan report

```
=================================================================
==141318==ERROR: AddressSanitizer: heap-buffer-overflow on address
0x602000000020 at pc 0x558cfb06a290 bp 0x7ffcfb418e70 sp 0x7ffcfb418e60
WRITE of size 4 at 0x602000000020 thread T0
    #0 0x558cfb06a28f in main /home/tang/st/lab6/heap.c:8
    #1 0x7fee6062a0b2 in __libc_start_main (/lib/x86_64-linux-
gnu/libc.so.6+0x240b2)
```

```
    #2 0x558cfb06a16d in _start (/home/tang/st/lab6/heap4+0x116d)

0x602000000020 is located 0 bytes to the right of 16-byte region
[0x602000000010,0x602000000020)
allocated by thread T0 here:
    #0 0x7fee60905808 in __interceptor_malloc
../../../../src/libsanitizer/asan/asan_malloc_linux.cc:144
    #1 0x558cfb06a24c in main /home/tang/st/lab6/heap.c:6
    #2 0x7fee6062a0b2 in __libc_start_main (/lib/x86_64-linux-
gnu/libc.so.6+0x240b2)

SUMMARY: AddressSanitizer: heap-buffer-overflow /home/tang/st/lab6/heap.c:8
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 00 00[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
  Shadow gap:              cc
==141318==ABORTING
```

Valgrind report

```
==141612== Memcheck, a memory error detector
==141612== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
```

```
==141612== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright
info
==141612== Command: ./heap5
==141612==
==141612== Invalid write of size 4
==141612==    at 0x109199: main (in /home/tang/st/lab6/heap5)
==141612==  Address 0x4a59050 is 0 bytes after a block of size 16 alloc'd
==141612==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-
gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==141612==    by 0x10918C: main (in /home/tang/st/lab6/heap5)
==141612==
==141612== Invalid read of size 4
==141612==    at 0x1091A7: main (in /home/tang/st/lab6/heap5)
==141612==  Address 0x4a59050 is 0 bytes after a block of size 16 alloc'd
==141612==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-
gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==141612==    by 0x10918C: main (in /home/tang/st/lab6/heap5)
==141612==
4==141612==
==141612== HEAP SUMMARY:
==141612==     in use at exit: 16 bytes in 1 blocks
==141612==   total heap usage: 2 allocs, 1 frees, 1,040 bytes allocated
==141612==
==141612== LEAK SUMMARY:
==141612==    definitely lost: 16 bytes in 1 blocks
==141612==    indirectly lost: 0 bytes in 0 blocks
==141612==      possibly lost: 0 bytes in 0 blocks
==141612==    still reachable: 0 bytes in 0 blocks
==141612==         suppressed: 0 bytes in 0 blocks
==141612== Rerun with --leak-check=full to see details of leaked memory
==141612==
==141612== For lists of detected and suppressed errors, rerun with: -s
==141612== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

ASan能，Valgrind能

Stack out-of-bounds read/write

Source code

```c
#include <stdio.h>

int main(){
    int a[100];
    int b = a[101];
    return 0;
}
```

ASan report

```
=================================================================
==146315==ERROR: AddressSanitizer: stack-buffer-overflow on address
0x7ffe56715f84 at pc 0x55e9aaa0530c bp 0x7ffe56715da0 sp 0x7ffe56715d90
READ of size 4 at 0x7ffe56715f84 thread T0
    #0 0x55e9aaa0530b in main /home/tang/st/lab6/stack.c:5
    #1 0x7fcb630770b2 in __libc_start_main (/lib/x86_64-linux-
gnu/libc.so.6+0x240b2)
    #2 0x55e9aaa0516d in _start (/home/tang/st/lab6/stack4+0x116d)

Address 0x7ffe56715f84 is located in stack of thread T0 at offset 452 in
frame
    #0 0x55e9aaa05238 in main /home/tang/st/lab6/stack.c:3

  This frame has 1 object(s):
    [48, 448) 'a' (line 4) <== Memory access at offset 452 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
/home/tang/st/lab6/stack.c:5 in main
Shadow bytes around the buggy address:
  0x10004acdaba0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10004acdabb0: 00 00 00 00 00 00 00 00 f1 f1 f1 f1 f1 f1 00 00
  0x10004acdabc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10004acdabd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10004acdabe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x10004acdabf0:[f3]f3 f3 f3 f3 f3 f3 f3 00 00 00 00 00 00 00 00
  0x10004acdac00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10004acdac10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10004acdac20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10004acdac30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x10004acdac40: 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
  Shadow gap:               cc
==146315==ABORTING
```

Valgrind report

```
==146905== Memcheck, a memory error detector
==146905== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==146905== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright
info
==146905== Command: ./stack
==146905==
==146905==
==146905== HEAP SUMMARY:
==146905==     in use at exit: 0 bytes in 0 blocks
==146905==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==146905==
==146905== All heap blocks were freed -- no leaks are possible
==146905==
==146905== For lists of detected and suppressed errors, rerun with: -s
==146905== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

ASan能，Valgrind不能

Global out-of-bounds read/write

Source code

```
#include <stdio.h>

int a[100] = {0};

int main(){
    printf("%d\n", a[101]);
    return 0;
}
```

ASan report

```
=================================================================
==142563==ERROR: AddressSanitizer: global-buffer-overflow on address
0x55f884ccf274 at pc 0x55f884ccc22b bp 0x7ffd7c591ca0 sp 0x7ffd7c591c90
READ of size 4 at 0x55f884ccf274 thread T0
    #0 0x55f884ccc22a in main /home/tang/st/lab6/global.c:6
    #1 0x7f2e69b9b0b2 in __libc_start_main (/lib/x86_64-linux-
gnu/libc.so.6+0x240b2)
    #2 0x55f884ccc12d in _start (/home/tang/st/lab6/global4+0x112d)
```

```
0x55f884ccf274 is located 4 bytes to the right of global variable 'a'
defined in 'global.c:3:5' (0x55f884ccf0e0) of size 400
SUMMARY: AddressSanitizer: global-buffer-overflow
/home/tang/st/lab6/global.c:6 in main
Shadow bytes around the buggy address:
  0x0abf90991df0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0abf90991e00: 00 00 00 00 00 00 00 00 f9 f9 f9 f9 f9 f9 f9 f9
  0x0abf90991e10: f9 f9 f9 f9 f9 f9 f9 f9 00 00 00 00 00 00 00 00
  0x0abf90991e20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0abf90991e30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0abf90991e40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00[f9]f9
  0x0abf90991e50: f9 f9 f9 f9 00 00 00 00 00 00 00 00 00 00 00 00
  0x0abf90991e60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0abf90991e70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0abf90991e80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0abf90991e90: 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
  Shadow gap:              cc
==142563==ABORTING
```

Valgrind report

```
==142975== Memcheck, a memory error detector
==142975== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==142975== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright
info
==142975== Command: ./global5
==142975==
0
==142975==
==142975== HEAP SUMMARY:
==142975==     in use at exit: 0 bytes in 0 blocks
==142975==   total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
```

```
==142975==
==142975== All heap blocks were freed -- no leaks are possible
==142975==
==142975== For lists of detected and suppressed errors, rerun with: -s
==142975== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

ASan能，Valgrind不能

Use-after-free

Source code

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    int *a = malloc(4 * sizeof(int));
    free(a);
    printf("%d\n", a[1]);
    return 0;
}
```

ASan report

```
=================================================================
==143467==ERROR: AddressSanitizer: heap-use-after-free on address
0x602000000014 at pc 0x55cf8623c28e bp 0x7ffd1efa9aa0 sp 0x7ffd1efa9a90
READ of size 4 at 0x602000000014 thread T0
    #0 0x55cf8623c28d in main /home/tang/st/lab6/use_after_free.c:7
    #1 0x7fadf9b9a0b2 in __libc_start_main (/lib/x86_64-linux-
gnu/libc.so.6+0x240b2)
    #2 0x55cf8623c16d in _start (/home/tang/st/lab6/use_after_free4+0x116d)

0x602000000014 is located 4 bytes inside of 16-byte region
[0x602000000010,0x602000000020)
freed by thread T0 here:
    #0 0x7fadf9e7540f in __interceptor_free
../../../../src/libsanitizer/asan/asan_malloc_linux.cc:122
    #1 0x55cf8623c24e in main /home/tang/st/lab6/use_after_free.c:6
    #2 0x7fadf9b9a0b2 in __libc_start_main (/lib/x86_64-linux-
gnu/libc.so.6+0x240b2)

previously allocated by thread T0 here:
    #0 0x7fadf9e75808 in __interceptor_malloc
../../../../src/libsanitizer/asan/asan_malloc_linux.cc:144
    #1 0x55cf8623c23e in main /home/tang/st/lab6/use_after_free.c:5
    #2 0x7fadf9b9a0b2 in __libc_start_main (/lib/x86_64-linux-
gnu/libc.so.6+0x240b2)
```

```
SUMMARY: AddressSanitizer: heap-use-after-free
/home/tang/st/lab6/use_after_free.c: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[fd]fd 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
  Shadow gap:              cc
==143467==ABORTING
```

Valgrind report

```
==143747== Memcheck, a memory error detector
==143747== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==143747== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright
info
==143747== Command: ./use_after_free5
==143747==
==143747== Invalid read of size 4
==143747==    at 0x1091B7: main (in /home/tang/st/lab6/use_after_free5)
==143747==  Address 0x4a59044 is 4 bytes inside a block of size 16 free'd
==143747==    at 0x483CA3F: free (in /usr/lib/x86_64-linux-
gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==143747==    by 0x1091AE: main (in /home/tang/st/lab6/use_after_free5)
==143747==  Block was alloc'd at
```

```
==143747==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-
gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==143747==    by 0x10919E: main (in /home/tang/st/lab6/use_after_free5)
==143747==
0
==143747==
==143747== HEAP SUMMARY:
==143747==     in use at exit: 0 bytes in 0 blocks
==143747==   total heap usage: 2 allocs, 2 frees, 1,040 bytes allocated
==143747==
==143747== All heap blocks were freed -- no leaks are possible
==143747==
==143747== For lists of detected and suppressed errors, rerun with: -s
==143747== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

ASan能，Valgrind能

Use-after-return

Sourece code

```
char* x;

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

int main() {

    foo();
    *x = 42; // Boom!

    return 0;
}
```

ASan report

```
$ gcc -fsanitize=address -g -o return use_after_return.c

$ ASAN_OPTIONS=detect_stack_use_after_return=1 ./return
```

執行前須加上 SAN_OPTIONS=detect_stack_use_after_return=1，才能抓到錯誤

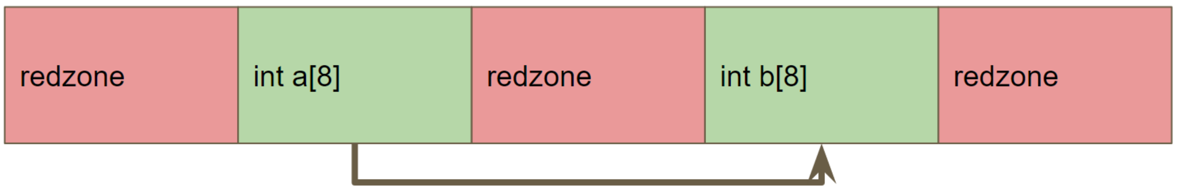
```
=================================================================
==147601==ERROR: AddressSanitizer: stack-use-after-return on address
0x7f17a7dfe03d at pc 0x56287cf9c32f bp 0x7ffdc9416910 sp 0x7ffdc9416900
WRITE of size 1 at 0x7f17a7dfe03d thread T0
    #0 0x56287cf9c32e in main /home/tang/st/lab6/use_after_return.c:13
    #1 0x7f17ab5240b2 in __libc_start_main (/lib/x86_64-linux-
gnu/libc.so.6+0x240b2)
    #2 0x56287cf9c10d in _start (/home/tang/st/lab6/return+0x110d)

Address 0x7f17a7dfe03d is located in stack of thread T0 at offset 61 in
frame
    #0 0x56287cf9c1d8 in foo /home/tang/st/lab6/use_after_return.c:5

  This frame has 1 object(s):
    [48, 90) 'stack_buffer' (line 6) <== Memory access at offset 61 is
inside 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-use-after-return
/home/tang/st/lab6/use_after_return.c:13 in main
Shadow bytes around the buggy address:
  0x0fe374fb7bb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe374fb7bc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe374fb7bd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe374fb7be0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe374fb7bf0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0fe374fb7c00: f5 f5 f5 f5 f5 f5 f5[f5]f5 f5 f5 f5 f5 f5 f5 f5
  0x0fe374fb7c10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe374fb7c20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe374fb7c30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe374fb7c40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0fe374fb7c50: 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
  Shadow gap:              cc
==147601==ABORTING
```

Valgrind report

```
==148050== Memcheck, a memory error detector
==148050== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==148050== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright
info
==148050== Command: ./return10
==148050==
==148050==
==148050== HEAP SUMMARY:
==148050==     in use at exit: 0 bytes in 0 blocks
==148050==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==148050==
==148050== All heap blocks were freed -- no leaks are possible
==148050==
==148050== For lists of detected and suppressed errors, rerun with: -s
==148050== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

ASan能，Valgrind不能

Part 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？



越過redzone

Source code

```
#include <stdio.h>

int main(){
    int a[8];
    int b[8];

    a[8+8] = 1;         //boom
    a[8+100] = 1;       //boom

    return 0;
}
```

```
$ gcc -fsanitize=address -g -o lab6_2 lab6_2.c
$ ./lab6_2
```

ASan 抓不到錯誤

```
a[8+0] ~ a[8+7]在redzone內可以抓到錯誤
a[8+8] 以後，越過redzone無法抓到錯誤
```