Name: Talise Baker-Matsuoka
ID: 7755662598
Email: bakermat@usc.edu

Brief program summary:

The goal of this extra credit section is to demonstrate a particular use after free error that ASAN can detect faster and more conveniently than GDB. I created my own example based on my knowledge of use after free errors because I thought it would be easier to create a simple example rather than find an example on the internet. The code with the error is shown in the file 7755662598extracredit_uaf.cpp.

Results:

Initial Error

When the code is run based on the incorrect code, the result is inconsistent and incorrect. Unfortunately, the program compiles and does not give any errors so this could be problematic.

```
bakermat@bakermat-QEMU-Virtual-Machine:~/ee355_lab6$ ./uaf.out
1831368798
bakermat@bakermat-QEMU-Virtual-Machine:~/ee355_lab6$ ./uaf.out
1534032867
```

Debugging with GDB

1. Compile and run as described in the how to run section
2. I tried adding a break in the main function so that the code would stop there and I could step through all the lines and stuff. After running the code as shown below, gdb does not tell me what may be wrong so the only thing I can do is examine the code line by line. This is not that much better than visually looking at my code, although I can see the value at each point

```
(gdb) run
Starting program: /home/bakermat/ee355_lab6/uaf.out
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/aarch64-linux-gnu/libthread_db.so.1".
-1431655741
[Inferior 1 (process 2958) exited normally]
(gdb) break main
Breakpoint 1 at 0xaaaaaaaa0a5c: file 7755662598extracredit_uaf.cpp, line 7.
(gdb)
```

3. Here, I use gdb commands to navigate through the code and check the value of the variable at each point. So far, everything is working properly.

```
(gdb) n
9               *ptr = 5;
(gdb) print ptr
$1 = (int *) 0xaaaaaaac3eb0
(gdb) print *ptr
$2 = 0
(gdb) n
12              delete ptr;
(gdb) print *ptr
$3 = 5
(gdb)
```

4. Continue going through line by line and I have now finally identified the line at which the error occurs. Now, I know that the issue is at line 12 in my code since after executing that line the issue appeared. So, now I can go back and fix that line, although I do not yet know the specific type of error. I can see here that the delete command is the issue.

```
(gdb) n
12              delete ptr;
(gdb) print *ptr
$3 = 5
(gdb) n
15              cout << *ptr << endl;
(gdb) p *ptr
$4 = -1431655741
(gdb) p ptr
$5 = (int *) 0xaaaaaaac3eb0
(gdb)
```

5. I can now implement a fix by either removing the command or removing any future references to the ptr variable since it has been deleted.

Debugging with ASAN

1. Compile and run the code as described in the how to run section yielding:

```
bakermat@bakermat-QEMU-Virtual-Machine:~/ee355_lab6$ g++ -fsanitize=address -ggdb -o uaf_asan 7755662598extracredit_uaf.cpp
bakermat@bakermat-QEMU-Virtual-Machine:~/ee355_lab6$ ./uaf_asan
=================================================================
==4057==ERROR: AddressSanitizer: heap-use-after-free on address 0xe06d388007b0 at pc 0xc65b9f6b0ee8 bp 0xffffff7f71210 sp 0xffffff7f71200
READ of size 4 at 0xe06d388007b0 thread T0
    #0 0xc65b9f6b0ee4 in main /home/bakermat/ee355_lab6/7755662598extracredit_uaf.cpp:15
    #1 0xe06d3cbc73fc in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58
    #2 0xe06d3cbc74d4 in __libc_start_main_impl ../csu/libc-start.c:392
    #3 0xc65b9f6b0d2c in _start (/home/bakermat/ee355_lab6/uaf_asan+0xd2c)

0xe06d388007b0 is located 0 bytes inside of 4-byte region [0xe06d388007b0,0xe06d388007b4)
freed by thread T0 here:
    #0 0xe06d3d101ecc in operator delete(void*, unsigned long) ../../../../src/libsanitizer/asan/asan_new_delete.cpp:172
    #1 0xc65b9f6b0e94 in main /home/bakermat/ee355_lab6/7755662598extracredit_uaf.cpp:12
    #2 0xe06d3cbc73fc in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58
    #3 0xe06d3cbc74d4 in __libc_start_main_impl ../csu/libc-start.c:392
    #4 0xc65b9f6b0d2c in _start (/home/bakermat/ee355_lab6/uaf_asan+0xd2c)

previously allocated by thread T0 here:
    #0 0xe06d3d100e0c in operator new(unsigned long) ../../../../src/libsanitizer/asan/asan_new_delete.cpp:99
    #1 0xc65b9f6b0e20 in main /home/bakermat/ee355_lab6/7755662598extracredit_uaf.cpp:7
    #2 0xe06d3cbc73fc in __libc_start_call_main ../sysdeps/nptl/libc_start_call_main.h:58
    #3 0xe06d3cbc74d4 in __libc_start_main_impl ../csu/libc-start.c:392
    #4 0xc65b9f6b0d2c in _start (/home/bakermat/ee355_lab6/uaf_asan+0xd2c)

SUMMARY: AddressSanitizer: heap-use-after-free /home/bakermat/ee355_lab6/7755662598extracredit_uaf.cpp:15 in main
Shadow bytes around the buggy address:
  0x1c1da71000a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c1da71000b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c1da71000c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c1da71000d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c1da71000e0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x1c1da71000f0: fa fa fa fa fa fa[fd]fa fa fa fa fa fa fa fa fa
  0x1c1da7100100: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c1da7100110: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c1da7100120: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c1da7100130: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x1c1da7100140: 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
==4057==ABORTING
bakermat@bakermat-QEMU-Virtual-Machine:~/ee355_lab6$
```

2. From this command, it is easy to identify that the issue is in line 15 of the file in the main function. If there were more functions, ASAN would be able to trace the issue through

multiple lines to the original source. ASAN identifies the issue as heap-use-after-free. Looking further also reveals that the delete operation is the issue. Now, the issue can be corrected.

3. Compile and run again to ensure there are no more errors. Since I wrote this code I know that there are none so this is unnecessary.

Why was ASAN better than GDB?

Here, GDB did not prove any more useful than careful, line by line examination of the code keeping track of what the value of each variable is at a given point. It did not help significantly to add any break points in the code since there were no infinite loops or crashes in the program. The program never even gave any warnings and GDB proved unhelpful beyond simple debugging.

On the other hand, ASAN detected the specific issue and identified it as use-after-free. It gave a specific line location in the code file where the issue occurred, and it identified the specific command that was causing the problem. In this case, ASAN required significantly fewer steps and gave a more definite answer regarding what the issue was. ASAN was faster because it required less commands and identified the error after running the compiled code. ASAN was more convenient since it identified the specific issue and line number and gave useful information about why the error was occurring. Thus, this code is an example of how ASAN can be faster and more convenient when dealing with use-after-free errors.

How to run:

Compile and run for GDB:
$ g++ -g 7755662598extracredit_uaf.cpp -o uaf.out
$ gdb uaf.out

Compile and run for ASAN:
$ g++ -fsanitize=address -ggdb -o uaf_asan 7755662598extracredit_uaf.cpp
$ ./uaf_asan

Running normally:
$ g++ -g 7755662598extracredit_uaf.cpp -o uaf.out
$ ./uaf.out

Resources:

- https://encyclopedia.kaspersky.com/glossary/use-after-free/
    - I used this resource to understand what a use-after-free error truly is and why it may occur. This resource goes into much more detail than is needed so I primarily focused on the paragraph here beginning with “Here’s how it happens.”