LAB 2

CS 361: Systems Programming / Spring 2023

Description

In this lab session, you will explore:

- The address sanitizer, which detects memory errors automatically
- Utility `strace`, which allows you to visualize the system calls made by processes
- Library `strtok`, which allows you to break a string given a delimiter

Please read this document carefully and follow the instructions on the last section to complete this lab session. When you answered all the questions, please show your work to the TA.

Address Sanitizer

The address sanitizer is an option available on modern compilers. When enabled, the address sanitizer tracks how your program uses memory. If your program uses memory incorrectly (e.g., read/write memory not allocated, read/write arrays out of bounds, free memory already freed earlier, etc.), the address sanitizer terminates your program with a descriptive error message. You can then use this error message to fix your program, and run it again.

Here is an example of a report issued by the address sanitizer:

```
=================================================================
==58000==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60b00000022c at pc 0x55a7345ca44b bp
0x7fff66b2dbb0 sp 0x7fff66b2dba0
WRITE of size 1 at 0x60b00000022c thread T0
    #0 0x55a7345ca44a in main file.c:361
    #1 0x7f037726128f  (/usr/lib/libc.so.6+0x2328f)
    #2 0x7f0377261349 in __libc_start_main (/usr/lib/libc.so.6+0x23349)
    #3 0x55a7345ca0e4 in _start ../sysdeps/x86_64/start.S:115

0x60b00000022c is located 36 bytes to the left of 100-byte region [0x60b000000250,0x60b0000002b4)
allocated by thread T0 here:
    #0 0x7f03774e4a89 in __interceptor_malloc /usr/src/debug/gcc/libsanitizer/asan/asan_malloc_linux.cpp:69
    #1 0x55a7345ca228 in main file.c:50
    #2 0x7f037726128f  (/usr/lib/libc.so.6+0x2328f)
```

In this example, the address sanitizer detected a buffer overflow on line 361 of file `file.c`. The sanitizer also reports that the memory was allocated on line 50 of file `file.c`.

The address sanitizer also detects segmentation faults:

```
AddressSanitizer:DEADLYSIGNAL
=================================================================
==59531==ERROR: AddressSanitizer: SEGV on unknown address 0x5615ee95a161 (pc 0x7f377ad16f0b bp
0x5615ee95a160 sp 0x7ffdf91fee80 T0)
==59531==The signal is caused by a WRITE memory access.
    #0 0x7f377ad16f0b in strtok_r (/usr/lib/libc.so.6+0x9df0b)
    #1 0x7f377af094f5 in __interceptor_strtok
/usr/src/debug/gcc/libsanitizer/sanitizer_common/sanitizer_common_interceptors.inc:658
    #2 0x5615ee9594f2 in file.c:41
    #3 0x7f377ac9c28f  (/usr/lib/libc.so.6+0x2328f)
    #4 0x7f377ac9c349 in __libc_start_main (/usr/lib/libc.so.6+0x23349)
    #5 0x5615ee9590e4 in _start ../sysdeps/x86_64/start.S:115
```

In this example, the program attempted to write outside of the memory allocated, which caused a segmentation fault. Looking at the report, we can see that this happened inside a function call to strtok on line 41 of file file.c.

`strace` utility

`strace` is a debugging program that tells you all of the system calls a program makes. System calls are requests a program makes to the operating system. Strace prints one syscall per line, and each line looks like this:

```
write(1, ``Hello, operating system'', 24) = 24
```

In this example, we can see that the program issued a write syscall, to the file-descriptor 1 (1st argument). We can see that the program is trying to write the string "Hello, operating system" (2nd argument), which is 24 bytes long (3rd argument). We can also see that the syscall returned 24, which in this case indicated that 24 were written to file-descriptor 1, as requested.

You can use `strace` on any program by using the command line: `strace <program>`

For instance, the command line `strace ./mystery1` will run `strace` on program mystery1.

`strace` has the following useful options:

- `strace -e write` will only show you the calls to write. (You can substitute your favorite system call for write.)
- `strace -f` will trace any processes spawned by the original process.
- `strace -p pid` will attach `strace` to the process ID of a running process.
- `strace -s NNN` will print the first NNN characters of any string (useful because `strace` truncates strings by default.)

`strtok` library

`strtok` is a library that breaks a string into "tokens", each token is delimited by another string. For instance, `strtok` can turn the string "Hello-this-is-an-example" into the tokens "Hello", "this", "is", "an", "example" when given the delimiter "-".

In this example, the first call to strtok would be:

```
char * tok;
char * string; // Contents are "Hello-this-is-an-example"
tok = strtok(string,"-");
```

At this point, variable tok points to the string "Hello". Note that `strtok` changes the original string, inserting a null character '\0' where the first '-' would be: "Hello\0this-is-an-example".

Future calls to strtok on the same string take NULL as the first argument:

```
tok = strtok(NULL, "-"); // tok is "this"
tok = strtok(NULL, "-"); // tok is "is"
tok = strtok(NULL, "-"); // tok is "an"
tok = strtok(NULL, "-"); // tok is "example"
tok = strtok(NULL, "-"); // tok is NULL
```

When `strtok` does not have any more tokens, it returns NULL to indicate the end of processing.

Guide

1. Accept the invitation for Lab 2 on Github classroom: https://classroom.github.com/a/LHXdwav-
2. Import the Github repository created to your machine using vscode, as explained in Assignment 0
3. Make sure that you can launch a terminal inside vscode via menus: Terminal > New Terminal
4. The repository has three files that you need: `lab2.c`, `mystery1`, `mystery2`.
5. Inspect the contents of source file `lab2.c`, type make to create the executable file lab2. Run it and answer Question 1 by inspecting the output of the address sanitizer.
6. Comment the line on `lab2.c` that causes the address sanitizer to issue an error.
7. Run programs mystery1 and mystery2 and observe what they output to the terminal.
8. Run `strace` on programs mystery1 and mystery2 and answer Questions 2 and 3.
9. Change program `lab2.c` to answer Question 4.
10. Show your work to the TA, commit and push your changes to the repo created in Step 2.

Questions

You can write the answers to your questions to file `ans.txt` in your repository, it then makes it easy to show your work to the TA.

1. What line of file lab2.c writes memory out of bounds?
2. Does `strace` truncate any string in mystery1? If so, what options can you pass to strace to see that string in its entirety?
3. What file does program mystery2 open? What does mystery2 write to that file? Note that strace may truncate strings.
4. In file `lab2.c`, how can you use `strtok` to generate the following output:

```
this is a string
this is another string
this is yet another string
*-* *--* *---* *----*
```

Extra / Optional

1. When using command `strace` on any executable (mystery1, mystery2, or lab2), there are a lot of syscalls before the program actually starts (e.g., before mystery1 outputs to the console using the write syscall to file-descriptor 1). What concept that we saw in class are those syscalls related to?

Grading

Show your UIC card to the TA when you enter the lab, or type your UIN on the chat when joining remotely. Stay in the session until you show your work, or until the TA announces that the lab is over.

- You have to remain present for the whole lab to get attendance, which you can then use to resubmit Assignment 2.
- You can leave early after showing your work to the TA (answers to all questions). In this case, you will get a 5% bonus in Assignment 2.