Sanitizer's & Git Tutorial

Alex Antonakakis, Stelios Malamas

University of Crete CSD

March 2024

Introduction

- Memory access bugs are common in C
- Sometimes they go unnoticed
- Sometimes they are obvious (e.g SEGV)
- Sanitizers can help eliminate and trace these bugs
- Available on most gcc and llvm versions
  - Address Sanitizer finds memory related bugs
  - Thread Sanitizer finds data races
  - UB Sanitizer finds undefined behaviour bugs

Address Sanitizer

- Traces memory access bugs and memory leaks
- Some common bugs include
  - Stack buffer overflow
  - Heap buffer overflow (e.g array overflow on malloc'ed memory)
  - Use after free
  - Memory leaks
- **Doesn't track uninitialized memory accesses!**
- Available via **-fsanitize=address** and **-fsanitize=leak**
- Makes the program about 2-3x slower, so don't use in production

Memory leak example

```
#include <stdlib.h>
int main(void) {
  void *p = malloc(1000);
}
```

Memory leak example

```
$ gcc main.c -g -fsanitize=address
$ ./a.out
```

```
=================================================================
==96076==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 1000 byte(s) in 1 object(s) allocated from:
    #0 0x7f250b6d9e8f in __interceptor_malloc
    ↪  ../../../../src/libsanitizer/asan/asan_malloc_linux.cp
    #1 0x55e2ba16f166 in main
    ↪  /home/ugrads/class20/csd4802/main.c:4
    #2 0x7f250b47fd09 in __libc_start_main
    ↪  ../csu/libc-start.c:308

SUMMARY: AddressSanitizer: 1000 byte(s) leaked in 1
↪  allocation(s).
```

Overflow example

```c
#include <stdlib.h>
int main(void) {
  int *p = malloc(1000 * sizeof(*p);

  p[1000] = 99; // off by one
}
```

Overflow example

```
$ gcc main.c -g -fsanitize=address
$ ./a.out
```

```
==96105==ERROR: AddressSanitizer: heap-buffer-overflow on
 ↪  address 0x619000001020 at pc 0x5623923e31bd bp
 ↪  0x7ffd7d01e7e0 sp 0x7ffd7d01e7d8
WRITE of size 4 at 0x619000001020 thread T0
    #0 0x5623923e31bc in main
     ↪  /home/ugrads/class20/csd4802/main.c:6
    #1 0x7f0295499d09 in __libc_start_main
     ↪  ../csu/libc-start.c:308
    #2 0x5623923e30a9 in _start
     ↪  (/home/ugrads/class20/csd4802/a.out+0x10a9)

Address 0x619000001020 is a wild pointer.
SUMMARY: AddressSanitizer: heap-buffer-overflow
 ↪  /home/ugrads/class20/csd4802/main.c:6 in main
```

Invalid free example

```c
#include <stdlib.h>
int main(void) {
  int *p = malloc(1000 * sizeof(*p);

  free(&p[5]);
}
```

Invalid free example

```
$ gcc main.c -g -fsanitize=address
$ ./a.out
```

```
==96119==ERROR: AddressSanitizer: attempting free on
↪  address which was not malloc()-ed: 0x619000000094 in
↪  thread T0
    #0 0x7fcac4058b6f in __interceptor_free
    ↪  ../../../../src/libsanitizer/asan/asan_malloc_linux.cp
    #1 0x55ccbc05a18a in main
    ↪  /home/ugrads/class20/csd4802/main.c:6
    #2 0x7fcac3dfed09 in __libc_start_main
    ↪  ../csu/libc-start.c:308
    #3 0x55ccbc05a0a9 in _start
    ↪  (/home/ugrads/class20/csd4802/a.out+0x10a9)

0x619000000094 is located 20 bytes inside of 1000-byte
↪  region [0x619000000080,0x619000000468)
```

Use after free example

```c
#include <stdlib.h>
int main(void) {
  int *p = malloc(1000 * sizeof(*p);

  free(p);

  p[5] = 98;
}
```

Use after free example

```
$ gcc main.c -g -fsanitize=address
$ ./a.out
```

```
==96161==ERROR: AddressSanitizer: heap-use-after-free on
 ↪  address 0x621000000114 at pc 0x55628d9871d6 bp
 ↪  0x7fffff103880 sp 0x7fffff103878
WRITE of size 4 at 0x621000000114 thread T0
    #0 0x55628d9871d5 in main
     ↪  /home/ugrads/class20/csd4802/main.c:8
    #1 0x7f0e2263ed09 in __libc_start_main
     ↪  ../csu/libc-start.c:308
    #2 0x55628d9870b9 in _start
     ↪  (/home/ugrads/class20/csd4802/a.out+0x10b9)
...
SUMMARY: AddressSanitizer: heap-use-after-free
 ↪  /home/ugrads/class20/csd4802/main.c:8 in main
```

Thread Sanitizer

- Traces data races and dead locks
- Available via -**fsanitize=thread**

Git-2

- Recognize git repos by the .git directory present
- **git config –get remote.origin.url** to see the repo source
- **git pull** is a good practice before starting to work on a collaborative project
  - Clone repo
  - Local changes must be committed or stashed

Branches (1/2)

- A branch represents an independent line of development
- Commits are performed separately in each branch. You can develop a feature without affecting your master branch
- **git branch ‘branch-name‘** Creating a new branch locally that is called ‘branch-name‘.
- **git push ‘remote-name‘ ‘branch-name‘** It pushes the branch called ‘branch-name‘ to the remote repo.
- **git branch** Returns a list of all the branches

Branches (2/2)

- **git checkout ‘branch-name‘** Changing our working branch to the ‘branch-name‘ branch
- Delete a Branch
  - **git branch -d ‘branch-name‘** Delete a branch that is called ‘branch-name‘ (-D to force delete)
  - **git push origin –delete ‘branch-name‘** Delete the branch called ‘branch-name‘ from the remote repo

Feature branch workflow example (1/3)

- All feature development happens in dedicated feature branches
- Master branch is always in a stable state
- Works well with pull/merge requests
- Steps
  - Start a new branch to work on something specific (new feature or known issue)
  - You can push your code to the central repository as often as you like since it won't affect any of your collaborators
  - You merge your branch into master once you are done

Feature branch workflow example (2/3)

- **git checkout master**
- **git pull**
- **git checkout -b new-feature**
- **git add file.c**
- **git commit -m "Implemented new feature"**

Feature branch workflow example (3/3)

- **git push –u origin new-feature**
- **git checkout master**
- **git pull**
- **git merge new-feature**
- **git push**

Tags

- **Tags are ref's that point to specific points in Git history**
- **Similar to a branch**
- **Useful in version control**

Creating a tag

- **git tag tagname**
- **git tag -a v1.4 (annotated tag)**
- **git tag -a v1.4 -m ”my version 1.4” (annotated with extra info)**

Pull requests

- **aka merge requests**
- **Merges changes in local forked repository to main repository**
- **Important when contributing to open-source projects**

Pull requests steps

- **Fork main repository**
- **Make changes locally**
- **Push changes to local repository**
- **Open a new pull request**
- **More on this on GitHub**