Семинар по C++ №7

Sanitizers and GDB

Вспоминаем проблемы

- Out of range
- Double free
- Memory leak
- Use after free
- Use after return
- ...

Sanitizers

- There are many of them:
- -fsanitize=address
- -fsanitize=pointer-compare
- -fsanitize=undefined
- ...

- The full list

Adress Sanitizer

- How to use?
  - g++ -fsanitize=address main.cpp
  - ./a.out
  - **ASAN_OPTIONS=detect_stack_use_after_return=1** ./a.out **–** for «use after return» checks
- Why to use?
  - index out of range
  - double free
  - use after free
  - memory leak
  - ...

Undefined

- How to use?
  - g++ -fsanitize=undefined main.cpp
  - ./a.out

- Why to use?
  - To detect undefined behaviour

How does it work?

- malloc/free wrappers
- memory access wrappers
- stack
- For more information, read GitHub

Before:

```
*address = ...;  // or: ... = *address;
```

After:

```
if (IsPoisoned(address)) {
  ReportError(address, kAccessSize, kIsWrite);
}
*address = ...;  // or: ... = *address;
```

Original code:

```
void foo() {
  char a[8];
  ...
  return;
}
```

Instrumented code:

```
void foo() {
  char redzone1[32];  // 32-byte aligned
  char a[8];          // 32-byte aligned
  char redzone2[24];
  char redzone3[32];  // 32-byte aligned
  int  *shadow_base = MemToShadow(redzone1);
  shadow_base[0] = 0xffffffff;  // poison redzone1
  shadow_base[1] = 0xffffff00;  // poison redzone2, unpoison 'a'
  shadow_base[2] = 0xffffffff;  // poison redzone3
  ...
  shadow_base[0] = shadow_base[1] = shadow_base[2] = 0; // unpoison all
  return;
}
```

GDB

- **g++ -g main.cpp** – compile with debug information
- **gdb a.out** – start debugging this executable
- **gdb --args a.out arg1 arg2** – pass command line arguments

- **break** (**b**) **func_name/line/...**– set a breakpoint
- **run** (**r**) – run a program to the end (or the nearest breakpoint)
- **continue** (**c**) – continue execution to the end (or the nearest breakpoint)

Navigation

- **step** (**s**) – go to the next line (step inside a function)
- **next** (**n**) – go to the next line (**do not** step inside a function)
- **stepi** / **nexti** – the same, but to the next assembly instruction
- **finish** – continue until the end of current function

Breakpoints

- **break** (**b**) **func_name/line/...–** set a breakpoint
- **info break** – get the list of breakpoints
- **delete *break_num* –** delete a breakpoint
- **clear –** delete all breakpoints
- **enable** / **disable** *break_num* – obvious

Stack

- **backtrace** (**bt**) – show backtrace
- **backtrace full** – show backtrace with local variables
- **list *func_name*/*line range*/...** – displays some code

utils

- **set var *var_name*=*value*** – set a value to a variable
- **p /format *var_name*** – display a variable
- Format
  - a – pointer
  - d – decimal
  - x – hexadecimal
  - f – floating point value
  - ...
- **gdb --pid *pid*** – attach to a working process

gdb with core dump

- /var/lib/systemd/coredump – path to core dump
- coredumpctl – cool utility for core dumps
- unlz4 – unpack core dump
- gdb a.out core – start core dump from gdb

TUI (Text User Interface) Mode

- tui enable/disable
- **Layouts:**

  layout next/prev/src/asm/split/regs
- **Focus:**
- **focus** next/prev/src/asm/regs/cmd

Sources

- GDB cheat sheet
- One more