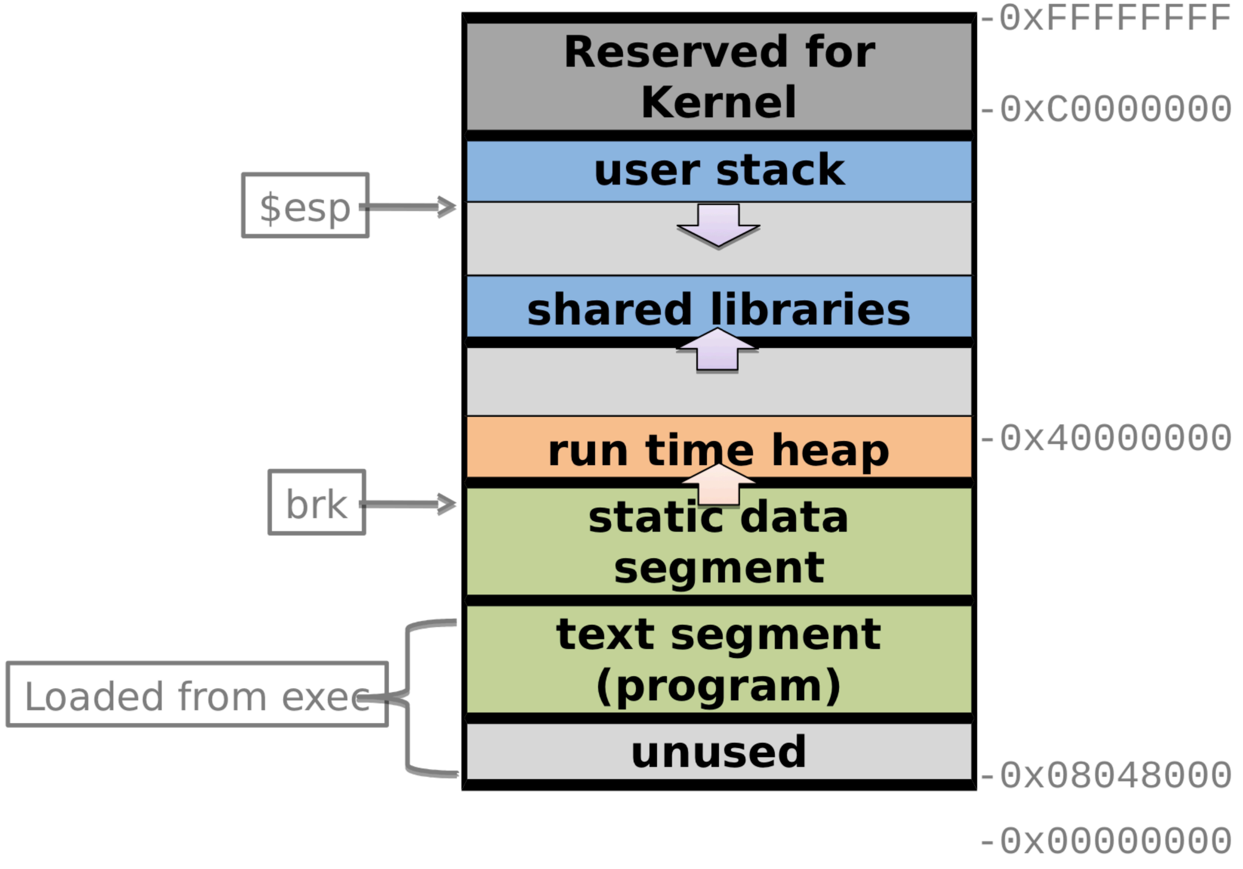
Securing Your Memory in a Valgrind Way

Wenjie Qiu

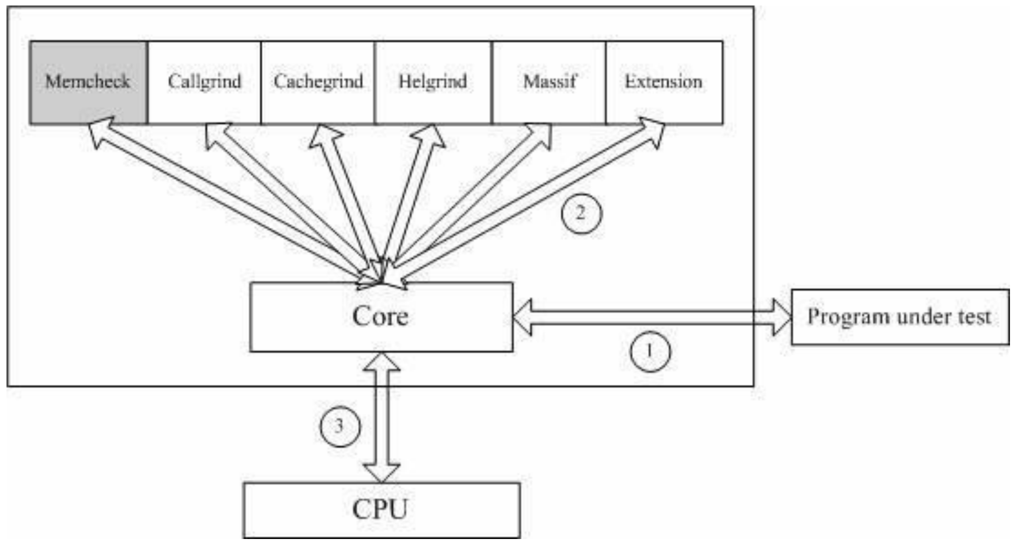
Content

- Memory Layout
- What is Valgrind
- Valgrind DEMO
- Why Valgrind
- Summary

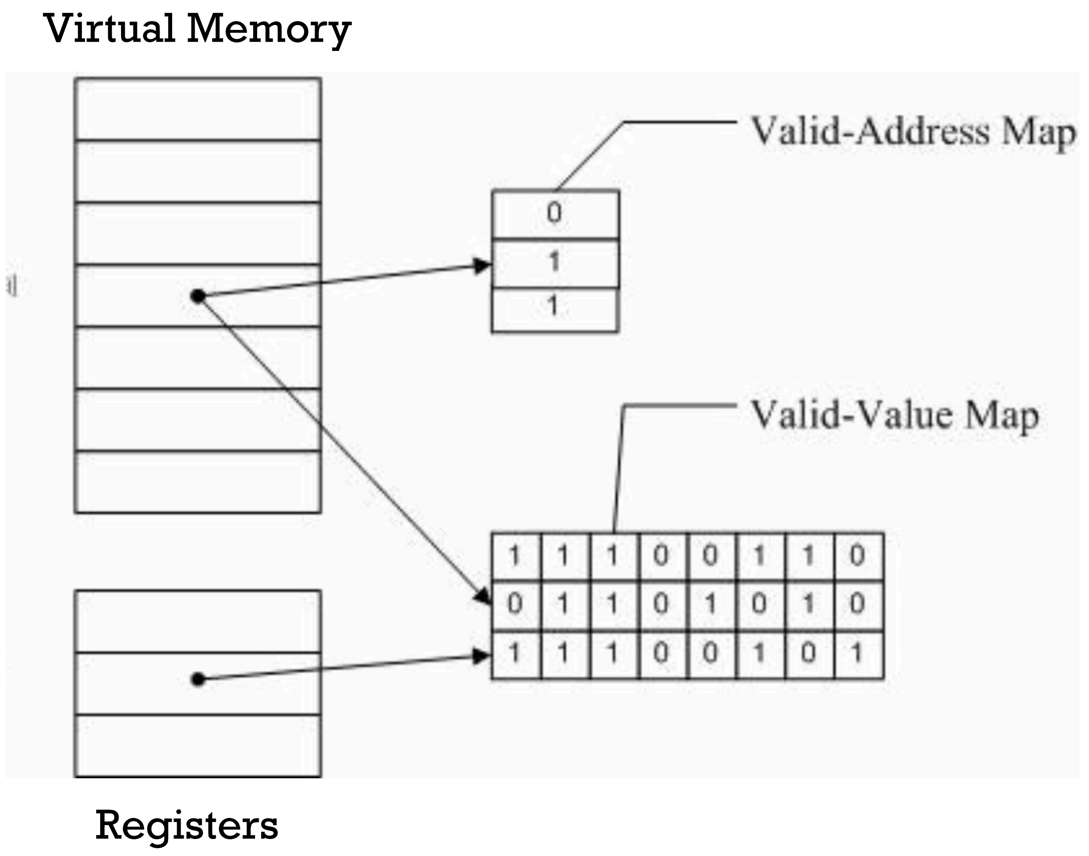
Linux (32-bit) process memory layout



What is Valgrind



- Valgrind is an instrumentation framework for building dynamic analysis tools.
- Can be used to detect memory error.
- Simulate a virtual CPU environment to perform the memory check.
- memcheck, callgrind, cachegrind, helgrind, massif, and extensions.
- Record every status, let me execute for you!



Valid-Address Map

- Each byte in VM → 1 bit in VA Map
- Valid for Read/Write?
- When touching invalid memory, report!

Valid-Value Map

- Each byte in VM → 8 bits in VV Map
- Each byte in Reg → 8 bits in VV Map
- Valid Value? (Initialized, not trash)
- When Address in Registers (may affect the program) contain trash, report!

Installation

- No support for the newest version of macOS (10.14)
- In Ubuntu 18.04:
- Build from source:

```
$ wget https://sourceware.org/pub/valgrind/valgrind-3.15.0.tar.bz2
$ tar -jxvf valgrind-3.15.0.tar.bz2 && cd valgrind-3.15.0
$ ./configure && make –j$(nproc)
$ sudo make install
```

- Package manager:

```
$ sudo apt install valgrind
```

- Integration with other IDE
  - Clion

Running

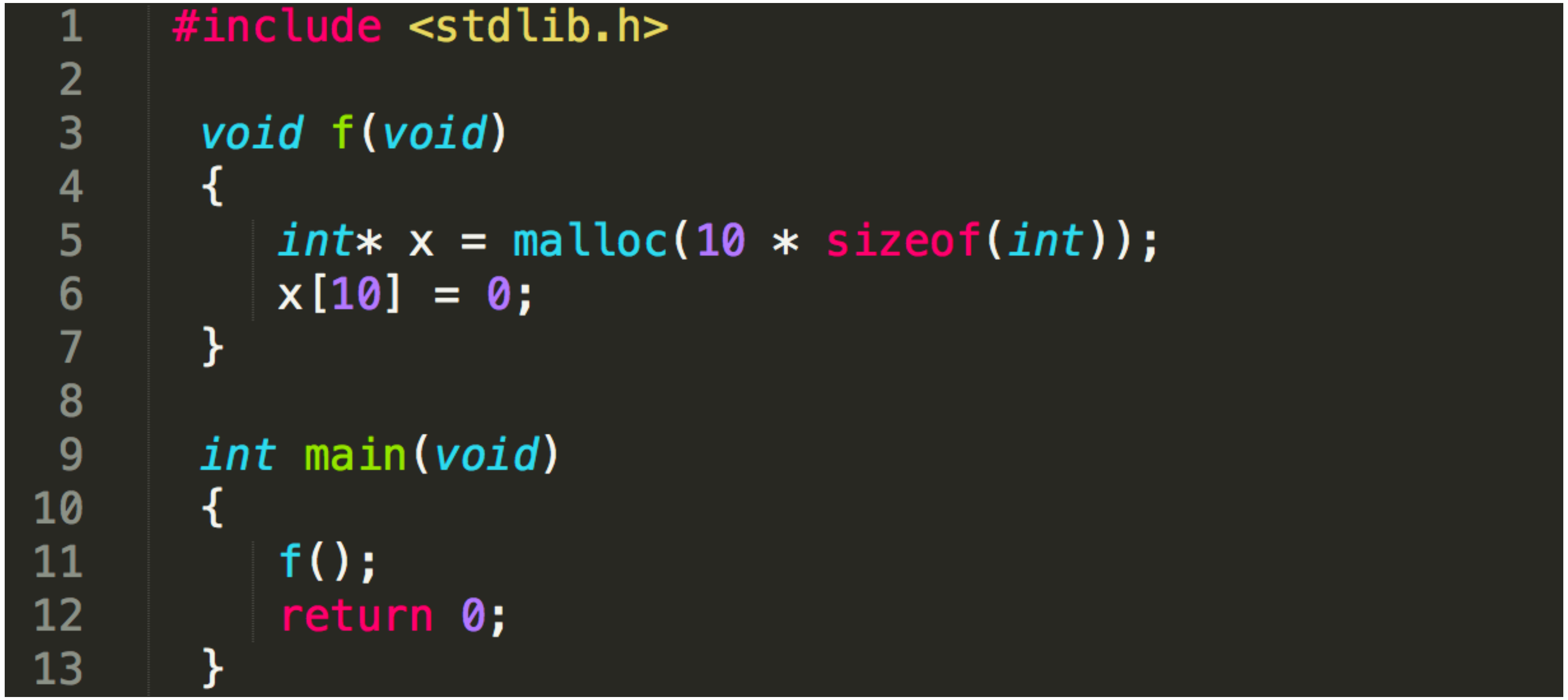
- In a normal way:

```
$ gcc -o valgrind_example valgrind_example.c
$ ./valgrind_example
```

- In a valgrind way:

```
$ gcc -g -O0 -o valgrind_example valgrind_example.c
$ valgrind --leak-check=full --log-file=leak.log ./valgrind_example
```

- Notice:
  - Using -g option will help valgrind to locate problems
  - Using -O0 optimization (no optimization) is recommended
  - Using -O1 if it's too slow for you!



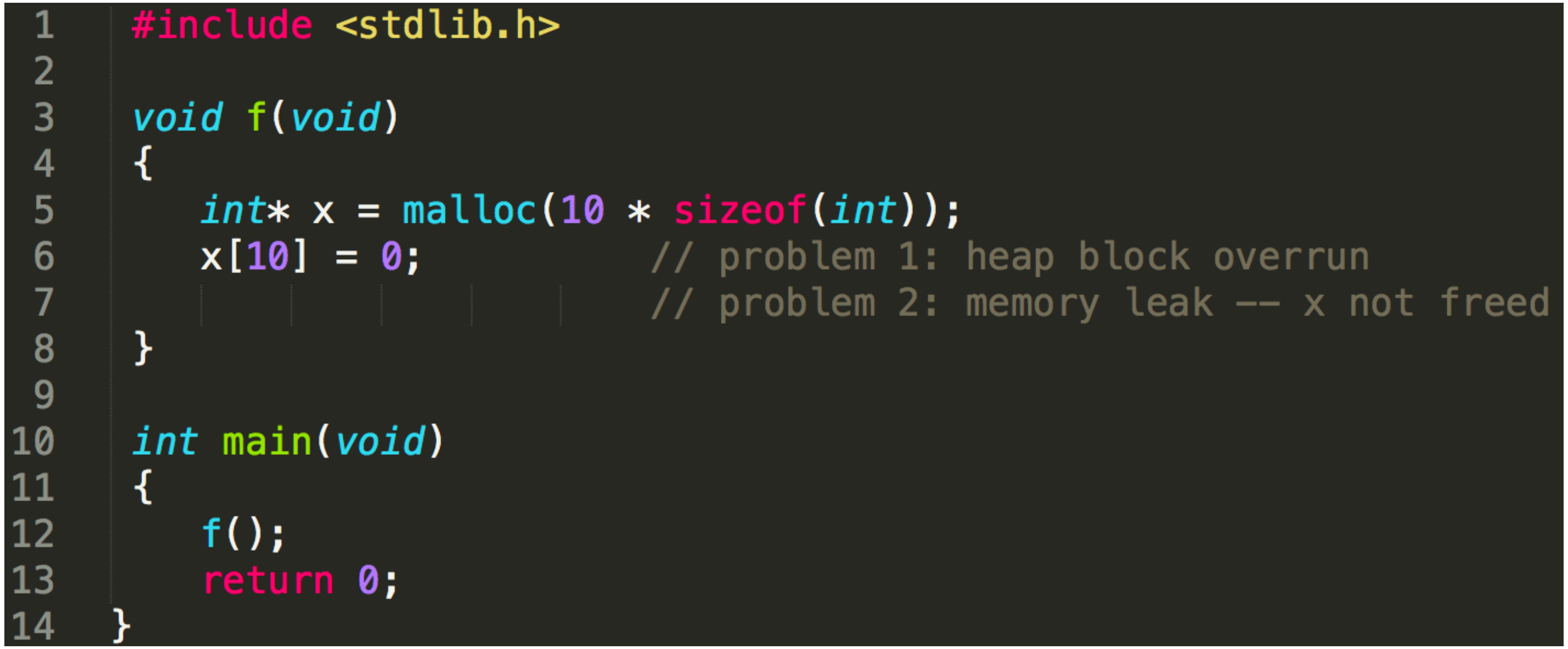
```
#include <stdlib.h>

 void f(void)
 {
     int* x = malloc(10 * sizeof(int));
     x[10] = 0;
 }

 int main(void)
 {
     f();
     return 0;
 }
```

DEMO

Where are the problems?



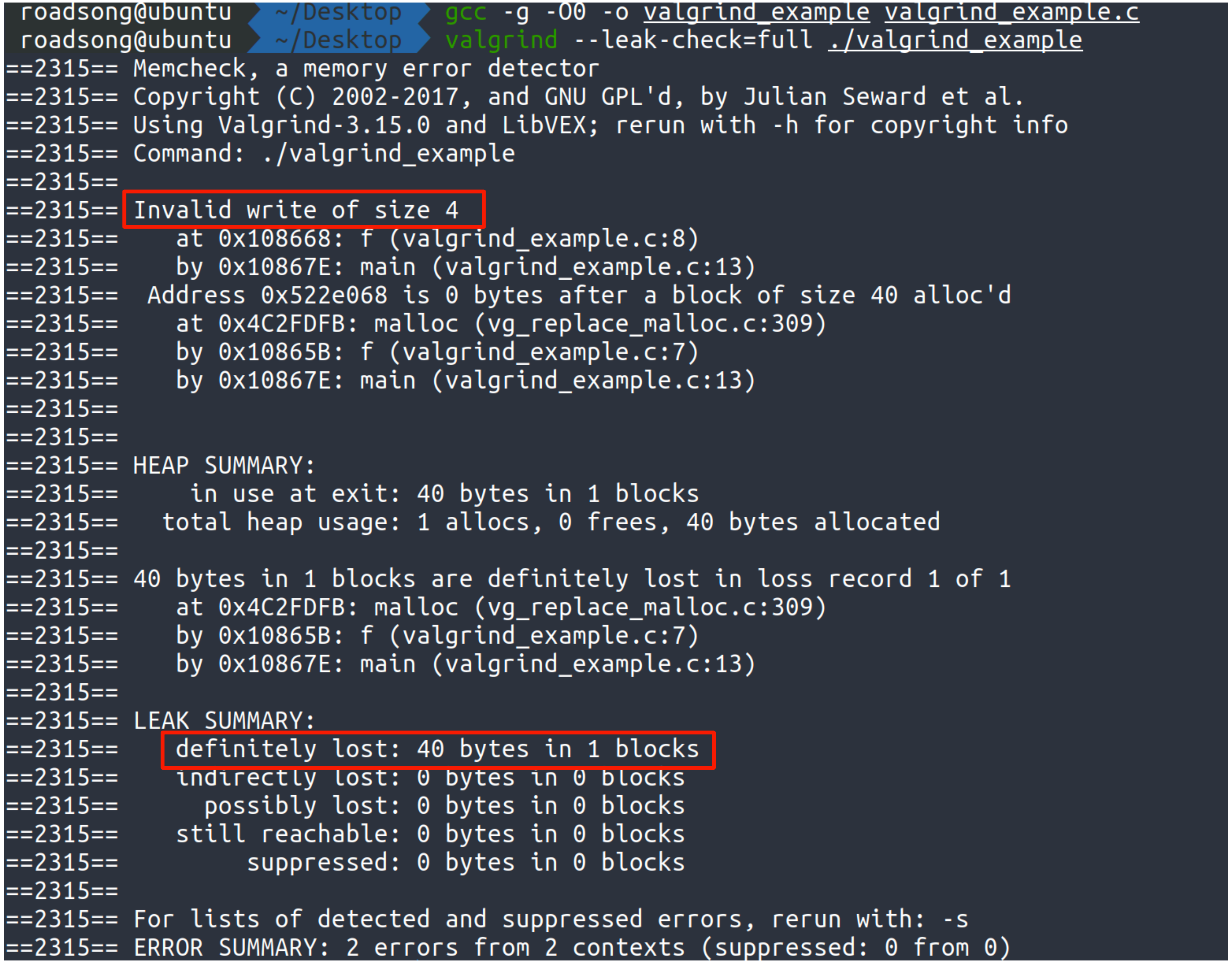
```c
#include <stdlib.h>

void f(void)
{
    int* x = malloc(10 * sizeof(int));
    x[10] = 0;          // problem 1: heap block overrun
                        // problem 2: memory leak -- x not freed
}

int main(void)
{
    f();
    return 0;
}
```

DEMO

How to fix?



```
 roadsong@ubuntu  ~/Desktop  gcc -g -O0 -o valgrind_example valgrind_example.c
 roadsong@ubuntu  ~/Desktop  valgrind --leak-check=full ./valgrind_example
==2315== Memcheck, a memory error detector
==2315== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2315== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==2315== Command: ./valgrind_example
==2315==
==2315== Invalid write of size 4
==2315==    at 0x108668: f (valgrind_example.c:8)
==2315==    by 0x10867E: main (valgrind_example.c:13)
==2315==  Address 0x522e068 is 0 bytes after a block of size 40 alloc'd
==2315==    at 0x4C2FDFB: malloc (vg_replace_malloc.c:309)
==2315==    by 0x10865B: f (valgrind_example.c:7)
==2315==    by 0x10867E: main (valgrind_example.c:13)
==2315==
==2315==
==2315== HEAP SUMMARY:
==2315==     in use at exit: 40 bytes in 1 blocks
==2315==   total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==2315==
==2315== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2315==    at 0x4C2FDFB: malloc (vg_replace_malloc.c:309)
==2315==    by 0x10865B: f (valgrind_example.c:7)
==2315==    by 0x10867E: main (valgrind_example.c:13)
==2315==
==2315== LEAK SUMMARY:
==2315==    definitely lost: 40 bytes in 1 blocks
==2315==    indirectly lost: 0 bytes in 0 blocks
==2315==      possibly lost: 0 bytes in 0 blocks
==2315==    still reachable: 0 bytes in 0 blocks
==2315==         suppressed: 0 bytes in 0 blocks
==2315==
==2315== For lists of detected and suppressed errors, rerun with: -s
==2315== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Lost in many ways!

- **definitely lost** means your program is leaking memory -- fix those leaks!
- **indirectly lost** means your program is leaking memory in a pointer-based structure. (E.g. if the root node of a binary tree is "definitely lost", all the children will be "indirectly lost".) If you fix the definitely lost leaks, the indirectly lost leaks should go away.
- **possibly lost** means your program is leaking memory, unless you're doing funny things with pointers. This is sometimes reasonable.
- **still reachable** means your program is probably ok -- it didn't free some memory it could have. This is quite common and often reasonable.
- **suppressed** means that a leak error has been suppressed. There are some suppressions in the default suppression files. You can ignore suppressed errors.

Two definition of memory leak

- The common one:

"Memory was allocated and was not subsequently freed before the program terminated (but at least we can find them)."

However, many programmers argue that certain types of memory leaks that fit this definition don't actually pose any sort of problem, and therefore should not be considered true "memory leaks". (But this is not a good habit!)

Still Reachable

- The stricter one:

"Memory was allocated and cannot be subsequently freed because the program no longer has any pointers to the allocated memory block (no way to find them)."

You cannot free memory that you no longer have any pointers to! Valgrind uses this stricter definition of the term "memory leak". This is the type of leak which can potentially cause significant heap depletion, especially for long lived processes.

Definitely Lost

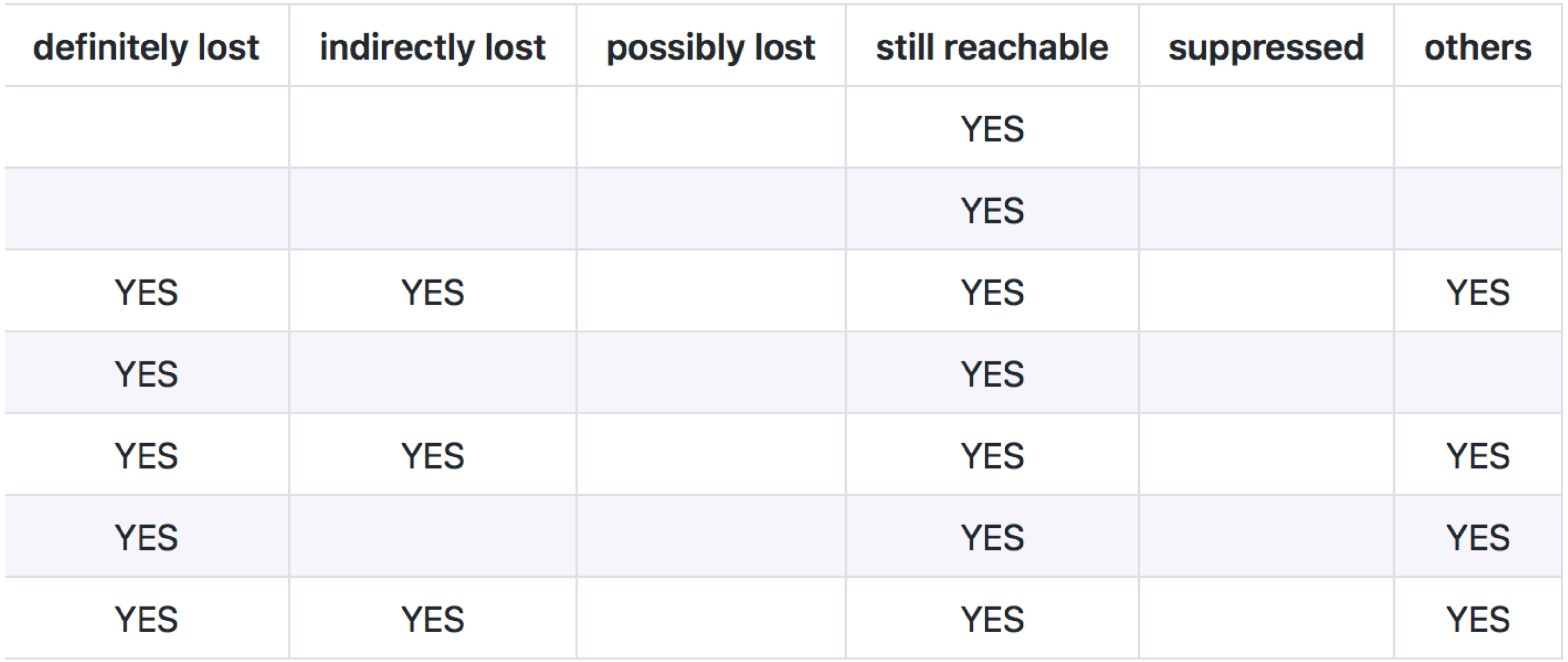
Why Valgrind

- In general, to find potential bugs!
- Such as memory leak, invalid read & write (and more)
- Why memory leak matters?
- What will happen if your server keep leaking memory?
- 7 * 24
- How much memory do you have?

Valgrind & IoT

- What if your IoT devices keep leaking memory?
- Relatively small memory -> Our of memory!
- Long live single process?
- Even have to solve “still reachable”

- To make things much more interesting…



definitely lost	indirectly lost	possibly lost	still reachable	suppressed	others
			YES		
			YES		
YES	YES		YES		YES
YES			YES		
YES	YES		YES		YES
YES			YES		YES
YES	YES		YES		YES

```
$ valgrind --leak-check=full --show-leak-kinds=all --log-file=[team_name].log ./embeddedlab
```

Valgrind Report

github.com/Roadsong/SecurityIoT-Memcheck

Summary

- Valgrind is an instrumentation framework for building dynamic analysis tools.
- Valgrind also has other tools other than memcheck!
- callgrind, cachegrind, helgrind, massif, and extensions.
- We also have other memory checking tools such as AddressSanitizer(ASAN), and gcc(>4.8) integrates it.

```
$ gcc [...] -fsanitize=address
```

- But Valgrind is a RUNTIME tool!

```
$ valgrind ./embeddedlab
```

Reference

- https://inst.eecs.berkeley.edu/~cs161/sp15/slides/lec3-sw-vulns.pdf
- http://valgrind.org
- http://valgrind.org/docs/valgrind2007.pdf
- https://www.ibm.com/developerworks/cn/linux/l-cn-valgrind/index.html
- https://github.com/google/sanitizers
- https://stackoverflow.com/questions/3840582/still-reachable-leak-detected-by-valgrind/3856938
- https://stackoverflow.com/questions/7886176/memory-not-freed-but-still-reachable-is-it-leaking

Thank you!