CS356: Program Analysis

W. Michael Petullo

University of Wisconsin–La Crosse

As of November 15, 2021

Program Analysis: grep

Use `grep` to look for:

- `gets`
- `strcpy` and `memcpy`
- `malloc`, `free`, and so on
- `printf(var)`
- Use of threads

`gets` is never safe, and the others are often abused.

Static Analysis (GCC): Use

GCC's `-fanalyzer` flag statically analyses a program to determine if it contains errors.

```
gcc -Wall -Wextra -fanalyzer foo.c -o foo
```

Important note: GCC might catch bugs, even if they rarely or do not manifest at runtime.

Code Analysis: A Segfault!

```
[user@host]$ gcc -g -o broken broken.c
[user@host]$ echo core >/proc/sys/kernel/core_pattern
[user@host]$ ./broken
Segmentation fault (core dumped)
[user@host]$ gdb broken core.N
(gdb) backtrace
```

Dynamic Analysis (Valgrind): Tools

A dynamic binary instrumentation framework; enables dynamic binary analysis tools:

Memcheck checks all userspace reads and writes of memory and intercepts all userspace allocations and frees. Memcheck will detect if your program:

- access memory it shouldn't,
- uses uninitialized values in dangerous ways,
- leaks memory,
- performs bad frees,
- or passes overlapping source and destination blocks to `memcpy`, etc.

Cachegrind is a cache profiler that helps pinpoint the source of cache misses.

Callgrind helps visualize cachegrind's findings.

Massif profiles a program's use of the heap.

Helgrind Identifies race conditions in multi-threaded programs.

Dynamic Analysis (Valgrind): Use

Valgrind will work with almost any binary program. However, you will receive more detailed output if the program bears debugging symbols. Here we compile using `gcc`'s `-g` option.

```
gcc -g foo.c -o foo
valgrind --leak-check=full ./foo
```

An example of a program that Valgrind cannot analyze is a self-modifying program.

Important note: Valgrind will only detect bugs that manifest at runtime. It will not reason about bugs that might manifest.

Program Analysis: Helgrind Use

Valgrind's Helgrind is a tool that aids in the analysis of running programs with respect to concurrency. Halgrind detects when a running program exhibits common concurrency errors, including misusing the POSIX thread API, data races, and applying inconsistent lock orderings.

```
gcc -g race.c -o race
valgrind --tool=helgrind ./race
```

Dynamic Analysis (Valgrind): Another Tool!

GCC's `-fsanitize` flag instruments memory access instructions to detect out-of-bounds and use-after-free errors.

```
gcc -Wall -Wextra -g -fsanitize=address foo.c -o foo
```

Important note: GCC's address sanitizer will only detect bugs the manifest at runtine. It will not reason about bugs that might manifest.

Memory Errors: Uninitialized Memory (uninit.c)

```c
#include <stdio.h>

void set_flag(int number, int *sign_flag) {
    if (number > 0) {
        *sign_flag = 1;
    } else if (number < 0) {
        *sign_flag = -1;
    }
}

int main(void) {
    int sign;
    set_flag(0, &sign);
    printf("%s\n", sign == 1 ? "positive" : "negative");
}
```

The CERT C Coding Standard 2nd ed.: MEM33-C

Memory Errors: Use After Free (use-after-free.c)

```
1  #include <stdlib.h>

3  struct node { int value; struct node *next; };

5  void free_list(struct node *head) {
6      for (struct node *p = head; p != NULL; p = p->next) {
7          free(p);
8      }
9  }

11 int main(void) {
12     struct node *n1 = calloc(1, sizeof(struct node));
13     struct node *n2 = calloc(1, sizeof(struct node));
14     n1->next = n2;
15     free_list(n1);
16 }
```

The CERT C Coding Standard 2nd ed.: MEM30-C

Memory Errors: Double Free (double-free.c)

```
1  #include <stdio.h>
2  #include <stdlib.h>

4  void f(char *p, size_t size) {
5      char *p2 = realloc(p, size);
6      if (p2 == NULL) {
7          free(p);
8      }
9  }

11 int main(void) {
12     char *p = malloc(BUFSIZ);
13     f(p, 0);
14 }
```

The CERT C Coding Standard 2nd ed.: MEM30-C

Memory Errors: Memory Leak (leak.c)

```
1  #include <stdio.h>
2  #include <stdlib.h>

4  int f(void) {
5      char *buf = malloc(BUFSIZ);
6      if (buf == NULL) {
7          return -1;
8      }
9      return 0;
10 }

12 int main(void)
13 {
14     f();
15 }
```

The CERT C Coding Standard 2nd ed.: MEM31-C

Memory Errors: Buffer Overflow (overflow.c)

```
1  #include <stdio.h>

3  void f() {
4      int c;
5      char *p, buf[16];
6      p = buf;
7      while ((c = getchar()) != '\n' && c != EOF) {
8          *p++ = (char) c;
9      }
10     *p++ = 0;
11 }

13 int main(void) {
14     f();
15 }
```

The CERT C Coding Standard 2nd ed.: MEM31-C

Program Analysis: Race Condition (race.c)

```
1  #include <pthread.h>

3  int var = 0;
4  void *fn(void *arg)
5  {
6      var++;
7      return NULL;
8  }

10 int main(void)
11 {
12     pthread_t p;
13     pthread_create(&p, NULL, fn, NULL);
14     var++;
15     pthread_join(p, NULL);
16 }
```

The CERT C Coding Standard 2nd ed.: CON43-C

Program Analysis: Assignments

Exam coming: in-class, written, one single-sided 8½×11 inch note sheet allowed. Note sheet must be hand-written with your hand, and you will turn it in with the exam.
No Möbius strips!

`https://www.flyn.org/courses/cs356/schedule`