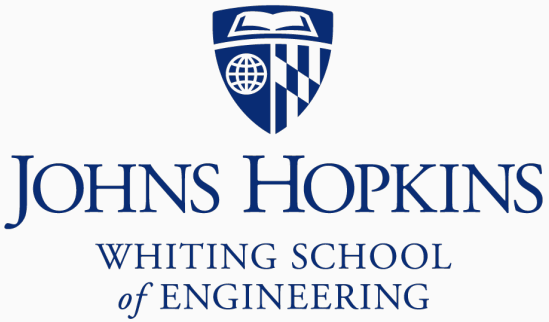
Valgrind

Ben Langmead

ben.langmead@gmail.com

www.langmead-lab.org



Source markdown available at github.com/BenLangmead/c-cpp-notes

Valgrind

Image from valgrind.org

Very easy-to-use tool for finding memory leaks and other pointer/memory mistakes

Compile your program with `-g` option for more helpful output from `valgrind`

```
valgrind --leak-check=full ./your-program <arg1> <arg2> ...
```

valgrind

From valgrind.org/docs/manual/faq.html:

> *The "grind" is pronounced with a short 'i' – ie. "grinned" (rhymes with "tinned") rather than "grined" (rhymes with "find"). Don't feel bad: almost everyone gets it wrong at first.*
>
> *Valgrind is the name of the main entrance to Valhalla (the Hall of the Chosen Slain in Asgard).*

valgrind

```c
#include <stdio.h>

int main() {
    printf("  ***  My program's output  ***\n");
    return 0;
}
```

valgrind

```
$ gcc -o valgrind_eg1 valgrind_eg1.c -std=c99 -pedantic -Wall -Wextra -g
$ valgrind --leak-check=full ./valgrind_eg1
  ***  My program's output  ***
==22== Memcheck, a memory error detector
==22== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==22== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==22== Command: ./valgrind_eg1
==22==
==22==
==22== HEAP SUMMARY:
==22==     in use at exit: 0 bytes in 0 blocks
==22==   total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
==22==
==22== All heap blocks were freed -- no leaks are possible
==22==
==22== For counts of detected and suppressed errors, rerun with: -v
==22== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

valgrind

Output of the program is interspersed with messages from `valgrind`

Some `valgrind` messages have to do with invalid reads and writes

- Usually, instances where we've dereferenced addresses not "belonging" to us

Everything from `HEAP SUMMARY` on has to do with memory leaks

- Failing to deallocate a pointer you allocated earlier

valgrind

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

char *string_copy(const char *orig) {
    char *fresh = malloc(strlen(orig) * sizeof(char));
    assert(fresh != NULL);
    strcpy(fresh, orig);
    return fresh;
}

int main() {
    char *hello_copy = string_copy("hello");
    assert(hello_copy != NULL);
    printf("%s\n", hello_copy);
    return 0;
}
```

valgrind

`valgrind` output indicates two problems:

"Invalid write" and "invalid read"

- We dereferenced addresses that didn't belong to us

valgrind

```
$ gcc -o buggy_strcpy buggy_strcpy.c -std=c99 -pedantic -Wall -Wextra -g
$ valgrind --leak-check=full ./buggy_strcpy
==21== Memcheck, a memory error detector
==21== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==21== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==21== Command: ./buggy_strcpy
==21==
==21== Invalid write of size 1
==21==    at 0x4C32C9D: strcpy (vg_replace_strmem.c:510)
==21==    by 0x40065D: string_copy (buggy_strcpy.c:9)
==21==    by 0x400675: main (buggy_strcpy.c:14)
==21==  Address 0x5221045 is 0 bytes after a block of size 5 alloc'd
==21==    at 0x4C2FB6B: malloc (vg_replace_malloc.c:299)
==21==    by 0x400626: string_copy (buggy_strcpy.c:7)
==21==    by 0x400675: main (buggy_strcpy.c:14)
==21==
==21== Invalid read of size 1
==21==    at 0x4C32B94: strlen (vg_replace_strmem.c:458)
==21==    by 0x4EB4D41: puts (in /usr/lib64/libc-2.26.so)
==21==    by 0x4006A5: main (buggy_strcpy.c:16)
==21==  Address 0x5221045 is 0 bytes after a block of size 5 alloc'd
==21==    at 0x4C2FB6B: malloc (vg_replace_malloc.c:299)
==21==    by 0x400626: string_copy (buggy_strcpy.c:7)
==21==    by 0x400675: main (buggy_strcpy.c:14)
```

valgrind

```
==21== HEAP SUMMARY:
==21==     in use at exit: 5 bytes in 1 blocks
==21==   total heap usage: 2 allocs, 1 frees, 4,101 bytes allocated
==21==
==21== 5 bytes in 1 blocks are definitely lost in loss record 1 of 1
==21==    at 0x4C2FB6B: malloc (vg_replace_malloc.c:299)
==21==    by 0x400626: string_copy (buggy_strcpy.c:7)
==21==    by 0x400675: main (buggy_strcpy.c:14)
==21==
==21== LEAK SUMMARY:
==21==    definitely lost: 5 bytes in 1 blocks
==21==    indirectly lost: 0 bytes in 0 blocks
==21==      possibly lost: 0 bytes in 0 blocks
==21==    still reachable: 0 bytes in 0 blocks
==21==         suppressed: 0 bytes in 0 blocks
==21==
==21== For counts of detected and suppressed errors, rerun with: -v
==21== ERROR SUMMARY: 3 errors from 3 contexts (suppressed: 0 from 0)
```

valgrind

Let's start with the "stack trace" for the memory leak:

```
==21== 5 bytes in 1 blocks are definitely lost in loss record 1 of 1
==21==    at 0x4C2FB6B: malloc (vg_replace_malloc.c:299)
==21==    by 0x400626: string_copy (buggy_strcpy.c:7)
==21==    by 0x400675: main (buggy_strcpy.c:14)
```

Look for the topmost function that's actually part of the code you wrote, and go to the file and line number indicated.

We wrote `main` & `string_copy`, but not `malloc`. `string_copy` is highest, so go to `buggy_strcpy.c:7`:

```
    char *fresh = malloc(strlen(orig) * sizeof(char));
```

valgrind

valgrind is saying that we fail to free the memory returned by this malloc

That's true! We should free it in main:

```
int main() {
    char *hello_copy = string_copy("hello");
    assert(hello_copy != NULL);
    printf("%s\n", hello_copy);
    free(hello_copy); // that's better
    return 0;
}
```

valgrind

```
==21== Invalid write of size 1
==21==    at 0x4C32C9D: strcpy (vg_replace_strmem.c:510)
==21==    by 0x40065D: string_copy (buggy_strcpy.c:9)
==21==    by 0x400675: main (buggy_strcpy.c:14)
==21==  Address 0x5221045 is 0 bytes after a block of size 5 alloc'd
==21==    at 0x4C2FB6B: malloc (vg_replace_malloc.c:299)
==21==    by 0x400626: string_copy (buggy_strcpy.c:7)
==21==    by 0x400675: main (buggy_strcpy.c:14)
```

Warning has two parts:

- Top stack trace: where “invalid write” happened
- Bottom: Where a nearby memory block was allocated; useful since mistake is usually that we go past the end of an allocated block

valgrind

```
char *string_copy(const char *orig) {
    // *** memory allocated on next line ***
    char *fresh = malloc(strlen(orig) * sizeof(char));
    assert(fresh != NULL);
    // *** invalid write on next line ***
    strcpy(fresh, orig);
    return fresh;
}
```

What's the mistake?

valgrind

`strlen` returns length of string *not counting* null terminator

But we need to `malloc` enough chars for string *and* terminator

valgrind

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

char *string_copy(const char *orig) {
    char *fresh = malloc((strlen(orig)+1) * sizeof(char)); // ** FIX 2
    assert(fresh != NULL);
    strcpy(fresh, orig);
    return fresh;
}

int main() {
    char *hello_copy = string_copy("hello");
    assert(hello_copy != NULL);
    printf("%s\n", hello_copy);
    free(hello_copy); // ** FIX 1
    return 0;
}
```

valgrind

Now let's look at the invalid read:

```
==21== Invalid read of size 1
==21==    at 0x4C32B94: strlen (vg_replace_strmem.c:458)
==21==    by 0x4EB4D41: puts (in /usr/lib64/libc-2.26.so)
==21==    by 0x4006A5: main (buggy_strcpy.c:16)
==21==  Address 0x5221045 is 0 bytes after a block of size 5 alloc'd
==21==    at 0x4C2FB6B: malloc (vg_replace_malloc.c:299)
==21==    by 0x400626: string_copy (buggy_strcpy.c:7)
==21==    by 0x400675: main (buggy_strcpy.c:14)
```

This is because the lack of null terminator causes the call to `printf` (which the compiler turned into a call to `puts`) to read beyond the end of `hello_copy`. We already fixed this.

valgrind

After fixes, we have a clean `valgrind` report:

```
$ gcc -o fixed_strcpy fixed_strcpy.c -std=c99 -pedantic -Wall -Wextra -g
$ valgrind --leak-check=full ./fixed_strcpy
hello
==34== Memcheck, a memory error detector
==34== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==34== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==34== Command: ./fixed_strcpy
==34==
==34==
==34== HEAP SUMMARY:
==34==     in use at exit: 0 bytes in 0 blocks
==34==   total heap usage: 2 allocs, 2 frees, 4,102 bytes allocated
==34==
==34== All heap blocks were freed -- no leaks are possible
==34==
==34== For counts of detected and suppressed errors, rerun with: -v
==34== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```