Chapter 61: Valgrind

Section 61.1: Bytes lost -- Forgetting to free

Here is a program that calls malloc but not free:

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

int main(int argc, char **argv)
{
    char *s;

    s = malloc(26); // the culprint

    return 0;
}
```

With no extra arguments, valgrind will not look for this error.

But if we turn on `--leak-check=yes` or `--tool=memcheck`, it will complain and display the lines responsible for those memory leaks if the program was compiled in debug mode:

```
$ valgrind -q --leak-check=yes ./missing_free
==4776== 26 bytes in 1 blocks are definitely lost in loss record 1 of 1
==4776==    at 0x4024F20: malloc (vg_replace_malloc.c:236)
==4776==    by 0x80483F8: main (missing_free.c:9)
==4776==
```

If the program is not compiled in debug mode (for example with the `-g` flag in GCC) it will still show us where the leak happened in terms of the relevant function, but not the lines.

This lets us go back and look at what block was allocated in that line and try to trace forward to see why it wasn't freed.

Section 61.2: Most common errors encountered while using Valgrind

Valgrind provides you with the *lines at which the error occurred* at the end of each line in the format `(file.c:line_no)`. Errors in valgrind are summarised in the following way:

```
ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

The most common errors include:

1. **Illegal read/write errors**

```
==8451== Invalid read of size 2
==8451==    at 0x4E7381D: getenv (getenv.c:84)
==8451==    by 0x4EB1559: __libc_message (libc_fatal.c:80)
==8451==    by 0x4F5256B: __fortify_fail (fortify_fail.c:37)
==8451==    by 0x4F5250F: __stack_chk_fail (stack_chk_fail.c:28)
==8451==    by 0x40059C: main (valg.c:10)
==8451==  Address 0x700000007 is not stack'd, malloc'd or (recently) free'd
```

This happens when the code starts to access memory which does not belong to the program. The size of the memory accessed also gives you an indication of what variable was used.

2. **Use of Uninitialized Variables**

```
==8795== 1 errors in context 5 of 8:
==8795== Conditional jump or move depends on uninitialised value(s)
==8795==    at 0x4E881AF: vfprintf (vfprintf.c:1631)
==8795==    by 0x4E8F898: printf (printf.c:33)
==8795==    by 0x400548: main (valg.c:7)
```

According to the error, at line 7 of the `main` of `valg.c`, the call to `printf()` passed an uninitialized variable to `printf`.

3. **Illegal freeing of Memory**

```
==8954== Invalid free() / delete / delete[] / realloc()
==8954==    at 0x4C2EDEB: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8954==    by 0x4005A8: main (valg.c:10)
==8954==  Address 0x5203040 is 0 bytes inside a block of size 240 free'd
==8954==    at 0x4C2EDEB: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8954==    by 0x40059C: main (valg.c:9)
==8954==  Block was alloc'd at
==8954==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==8954==    by 0x40058C: main (valg.c:7)
```

According to valgrind, the code freed the memory illegally (a second time) at *line 10* of `valg.c`, whereas it was already freed at *line 9*, and the block itself was allocated memory at *line 7*.

Section 61.3: Running Valgrind

```
valgrind ./my-program arg1 arg2 < test-input
```

This will run your program and produce a report of any allocations and de-allocations it did. It will also warn you about common errors like using uninitialized memory, dereferencing pointers to strange places, writing off the end of blocks allocated using malloc, or failing to free blocks.

Section 61.4: Adding flags

You can also turn on more tests, such as:

```
valgrind -q --tool=memcheck --leak-check=yes ./my-program arg1 arg2 < test-input
```

See valgrind --help for more information about the (many) options, or look at the documentation at http://valgrind.org/ for detailed information about what the output means.