CSc 352

Valgrind

Russell Lewis (sub for Benjamin Dicken)

Announcements

- Ben is on paternity leave
  - Back in 3 weeks (Apr 4)
- New Lecture form:
  - 50 minutes in class
  - 25 minutes by video (posted next day)
    - https://www.youtube.com/playlist?list=PL-F3IhGTDSSqe5cMDqrLdHkG0bleuA_xq
  - Slides posted on D2L
- My 352 Office Hours: 11am-noon, MWF
- Mask mandate changing next week (21 Mar)

Recap:

- **`void* malloc(size_t size);`**
  - Allocates **size** bytes and returns the pointer to it, or NULL if failed to alloc
- **`void* calloc(size_t n_items, size_t size);`**
  - Allocates **(n_items* size)** bytes and returns the pointer to it, or NULL if failed to alloc
- **`void free(void * ptr);`**
  - Frees the memory pointer to by **ptr** so that your program can no longer rely on having access to that memory

Bugs!

The program on the next slide is buggy.

- Q1: What is the bug?
- Q2: Will you notice the bug (most times)? Why or why not?

Bugs!

```c
int main()
{
    int *buf = malloc(3*sizeof(int));

    buf[0] = 123;
    buf[1] = 456;
    buf[2] = 789;
    printf("%d %d %d\n",
           buf[0],buf[1],buf[2]);
    printf("\n");

    free(buf);
```

```c
    printf("%d\n", buf[0]);
    printf("%d\n", buf[1]);
    printf("%d\n", buf[2]);
    printf("\n");

    buf[0] = 999999;
    printf("%d\n", buf[0]);
    printf("%d\n", buf[1]);
    printf("%d\n", buf[2]);

    return 0;
}
```

Bugs! (solution)

- Data that you have `free()`d no longer belongs to you!
  - Must never attempt to use a buffer after you've freed it

- Will you notice the bug? Maybe.
  - `free()` may leave the buffer untouched
  - Or, it can change the contents
  - Might give to another thread (if multithreaded)
  - Debug tools might gripe

De-Bug!

- Can we debug the program without digging deep into the source code?

- Run it with valgrind:

```
valgrind ./my_program_name
```

- What debug information does it give?

```
russelll@lectura:~$ valgrind ./lec_13_valgrind_act
==4076377== Memcheck, a memory error detector
==4076377== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==4076377== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==4076377== Command: ./foo
==4076377==
123
456
789


... output continues on next slide ...
```

- Your program does its ordinary work.

- `valgrind` tells you that it is running.

```
... continued ...

==4076377== Invalid read of size 4
==4076377==    at 0x109233: main (in /home/russelll/foo)
==4076377==  Address 0x4a5b040 is 0 bytes inside a block of size 8 free'd
==4076377==    at 0x483CA3F: free (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==4076377==    by 0x109222: main (in /home/russelll/foo)
==4076377==  Block was alloc'd at
==4076377==    at 0x483B7F3: malloc (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==4076377==    by 0x1091BE: main (in /home/russelll/foo)
==4076377==
x: 123

... output continues on next slide ...
```

- `valgrind` has detected a runtime error. It is telling you that you are reading something you no longer own.

```
... continued ...

==4076377== Invalid read of size 4
==4076377==    at 0x109233: main (in /home/russelll/foo)
==4076377==  Address 0x4a5b040 is 0 bytes inside a block of size 8 free'd
==4076377==    at 0x483CA3F: free (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==4076377==    by 0x109222: main (in /home/russelll/foo)
==4076377==  Block was alloc'd at
==4076377==    at 0x483B7F3: malloc (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==4076377==    by 0x1091BE: main (in /home/russelll/foo)
==4076377==
x: 123

... output continues on next slide ...
```

- This tells you where in your code the error occurred.
- We'll get more details soon.

```
... continued ...

==4076377== Invalid read of size 4
==4076377==    at 0x109233: main (in /home/russelll/foo)
==4076377==  Address 0x4a5b040 is 0 bytes inside a block of size 8 free'd
==4076377==    at 0x483CA3F: free (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==4076377==    by 0x109222: main (in /home/russelll/foo)
==4076377==  Block was alloc'd at
==4076377==    at 0x483B7F3: malloc (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==4076377==    by 0x1091BE: main (in /home/
==4076377==
x: 123

... output continues on next slide ...
```

- This tells you what pointer you used, and why it is invalid. Notice that it tells us that we are at the very start of a free'd block.

```
... continued ...

==4076377== Invalid read of size 4
==4076377==    at 0x109233: main (in /home/russelll/foo)
==4076377==  Address 0x4a5b040 is 0 bytes inside a block of size 8 free'd
==4076377==    at 0x483CA3F: free (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==4076377==    by 0x109222: main (in /home/russelll/foo)
==4076377==  Block was alloc'd at
==4076377==    at 0x483B7F3: malloc (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==4076377==    by 0x1091BE: main (in /home/russelll/foo)
==4076377==
x: 123

... output continues on next slide ...
```

- This tells us what code free()d the block.

```
... continued ...

==4076377== Invalid read of size 4
==4076377==    at 0x109233: main (in /home/russelll/foo)
==4076377==  Address 0x4a5b040 is 0 bytes inside a block of size 8 free'd
==4076377==    at 0x483CA3F: free (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==4076377==    by 0x109222: main (in /home/russelll/foo)
==4076377==  Block was alloc'd at
==4076377==    at 0x483B7F3: malloc (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==4076377==    by 0x1091BE: main (in /home/russelll/foo)
==4076377==
123

... output continues on next slide ...
```

- This tells us what code malloc()d the block.

```
... continued ...

==4076377== Invalid read of size 4
==4076377==    at 0x109233: main (in /home/russelll/foo)
==4076377==  Address 0x4a5b040 is 0 bytes inside a block of size 8 free'd
==4076377==    at 0x483CA3F: free (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==4076377==    by 0x109222: main (in /home/russelll/foo)
==4076377==  Block was alloc'd at
==4076377==    at 0x483B7F3: malloc (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==4076377==    by 0x1091BE: main (in /home/russelll/foo)
==4076377==
123

... output continues on next slide ...
```

- `valgrind` will allow your program to continue running. So we go ahead and print what we read.

```
... continued ...

==4076377== Invalid read of size 4
==4076377==    at 0x10924C: main (in /home/russelll/foo)
==4076377==  Address 0x4a5b044 is 4 bytes inside a block of size 8 free'd
==4076377==    at 0x483CA3F: free (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==4076377==    by 0x109222: main (in /home/russelll/foo)
==4076377==  Block was alloc'd at
==4076377==    at 0x483B7F3: malloc (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==4076377==    by 0x1091BE: main (in /home/russelll/foo)
==4076377==
456

... output continues on next slide ...
```

- `valgrind` will report each error it finds. We read three fields from the free()d block, so we get three errors.

```
... continued ...

==175506== Invalid write of size 4
==175506==    at 0x10928D: main (lec_13_valgrind_act1.c:22)
==175506==  Address 0x4a5b040 is 0 bytes inside a block of size 12 free'd
==175506==    at 0x483CA3F: free (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==175506==    by 0x10922B: main (lec_13_valgrind_act1.c:15)
==175506==  Block was alloc'd at
==175506==    at 0x483B7F3: malloc (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==175506==    by 0x1091BE: main (lec_13_valgrind_act1.c:7)
==175506==

... output continues on next slide ...
```

- `valgrind` will also tell us if we write to buffers that we've free()d

```
... continued ...

==4076377==
==4076377== HEAP SUMMARY:
==4076377==     in use at exit: 0 bytes in 0 blocks
==4076377==   total heap usage: 2 allocs, 2 frees, 1,032 bytes allocated
==4076377==
==4076377== All heap blocks were freed -- no leaks are possible
==4076377==
==4076377== For lists of detected and suppressed errors, rerun with: -s
==4076377== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
russelll@lectura:~$
```

- `valgrind` gives a final report. We free()d all of the memory we malloc()d, but we had two errors along the way.

Better Debug Data

- `valgrind` was able to tell us what function we were running, but we'd like more details.

- We can turn on debugging information as necessary

```
gcc -Wall -Werror -std=c11 -g foo.c -o foo
```

- Typically not turned on for "production" builds
  - May slow down your program (though not a lot)
  - Will make the executable larger
  - May give hackers more insight into your program

Better Debug Data

- Debug data is useful to a variety of tools
  - `gdb`
  - `valgrind`
  - Perhaps useful if debugging a crash report

```
==4078464== Invalid read of size 4
==4078464==    at 0x109233: main  (foo.c:25)
==4078464==  Address 0x4a5b040 is 0 bytes inside a block of size 8 free'd
==4078464==    at 0x483CA3F: free (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==4078464==    by 0x109222: main  (foo.c:22)
==4078464==  Block was alloc'd at
==4078464==    at 0x483B7F3: malloc (in
/usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==4078464==    by 0x1091BE: main  (foo.c:14)
==4078464==
x: 123
```

- `valgrind` can give you exact line numbers in its bug reports.

Brainstorm!

- Talk with your neighbors nearby. Let's see if we can brainstorm lots and lots of possible memory-related errors. Then, we'll see how many of them `valgrind` can help us find.
- Each group should come up with at least 5 different errors to check for. Be creative! Can you think of the weirdest errors?
    - malloc() related
    - Stack related
    - Constants
- For each error, think about how to write a small program which would give an example of this error, and we'll try to test it using `valgrind`.

My Ideas (page 1)

- Read/write NULL

- Random pointer (reads, writes)
- Read/write before or after a malloc() buffer
- Read/write before or after a buffer on the stack
- Read/write buffer (on stack) returned by another function

- strlen() on malloc() buffer, full of non-zero data
- strcpy() that overruns malloc() buffer
- strcpy() where src,dest overlap each other

My Ideas (page 2)

- Read of uninitialized malloc() space
- Read of uninitialized stack space

- Write to string constant

- ~~Use data after free() (reads, writes)~~ *already done*
- Double free()
- Never free()
- free() wrong address (inside buf, outside buf, globals, stack)

My Ideas (page 3)

- ~~Use data after free() (reads, writes)~~ *already done*
- Double free()
- Never free()
- free() wrong address (inside buf, outside buf, globals, stack)

- realloc() of wild pointer
- realloc() of free()d pointer
- realloc() of offset *into* a current buffer
- realloc() of stack memory, or globals

Test It Out

- I've created a Google Doc for the entire class to share; see D2L

- Divide into small groups
  - Each group takes one of the problems (your ideas, or mine)
  - Write a small program that gives an example of the bug
  - Compile & run it
    - Is the error obvious, or silent? Can you explain why?
    - If you run under `valgrind`, does that change?
  - Drop your code (with a summary of your results) into the Doc

- We'll review examples as a class, once they are ready