15-213 Recitation: C Review

TA's

19 Feb 2018

Agenda

- Logistics
- Attack Lab Conclusion
- C Activities
- C Programming Style
- C Exercise
- Appendix:
  - Valgrind
  - Clang / LLVM
  - Cache Structure

Logistics

- Attack Lab is due **tomorrow at midnight**!
  - Come to office hours for help
  - `rtarget` phase 5 is only worth 5 points
    - 0.2% of your grade ≈ 0% of your grade
- Cache Lab will be released shortly thereafter!

Attack Lab Conclusion

- Don’t use functions vulnerable to buffer overflow (like `gets`)
  - Use functions that allow you to specify buffer lengths:
    - `fgets` instead of `gets`
    - `strncpy` instead of `strcpy`
    - `strncat` instead of `strcat`
    - `snprintf` instead of `sprint`
  - Use `sscanf` and `fscanf` with input lengths (`%213s`)

- Stack protection makes buffer overflow very hard…
  - But very hard ≠ impossible!

C Activities

- Basic C Programming Questions
  - Activity 1 and 2: Common Pointer Mistakes
  - Activity 3 and 4: Common Malloc Mistakes
  - Activity 5: Common Macro Mistakes
- Learn to use `getopt`
  - Extremely useful for Cache Lab
  - Processes command line arguments

C Activities

- Pair up!
  - Login to a shark machine

    $ wget http://www.cs.cmu.edu/~213/activities/rec5.tar

    $ tar xvf rec5.tar

    $ cd rec5

    $ make

- Open act_pointers.c

C Activity (pointers): act1

- Open act_pointers.c, make sure main is running act1
- $ make pointers
- $ ./act_pointers
- What happened? Let's try to debug
- $ gdb act_pointers
- (gdb) run
- (gdb) backtrace
- $ valgrind --leak-check=full ./act_pointers
- How would you fix this?
- Is there another way?

C Activity (pointers): act2

- Switch main to run act2
- $ make pointers
- $ ./act_pointers
- What happened? Let’s try to debug
- $ gdb act_pointers
- (gdb) break act2
- (gdb) run
- (gdb) list
- (gdb) watch *d
- (gdb) continue

C Activity (malloc): act3

- Make sure main is running act3
- $ make malloc
- $ ./act_malloc
- What happened? Let’s try to debug
- $ valgrind ./act_malloc
- Are there any errors?
- Is there any memory lost?

C Activity (malloc): act4

- Switch main to run act4
- $ make malloc
- $ ./act_malloc
- What happened? Let’s try to debug
- $ gdb ./act_malloc
- (gdb) run
- (gdb) backtrace

C Activity (macros): act5

- Switch to act_macros.c
- $ make macros
- $ ./act_macros
- What happened? Let's try to debug
- $ gcc -std=c99 -E act_macros.c > expanded.txt
- Open expanded.txt and look at act5
- How did the macros expand?

C Activity (macros): Review

- Macros are good for compile-time decisions
  - Assert, requires, etc
  - dbg_print
- Macros are not functions and should not be used interchangeably
- Use functions whenever you can

C Activities Conclusion

- Did you answer every question correctly? If not…
  - Refer the C Bootcamp slides
- Was the test so easy you were bored? If not…
  - Refer the C Bootcamp slides
- When in doubt…
  - Refer the C Bootcamp slides
- This will be *very* important for the rest of this class, so make sure you are comfortable with the material covered or come to the C Bootcamp!

C Programming Style

- Document your code with comments
- Check error and failure conditions
- Write modular code
- Use consistent formatting
- Avoid memory and file descriptor leaks

- Warning: *Dr. Evil* has returned to grade style on Cache Lab! ᨏ
  - Refer to full 213 Style Guide: http://cs.cmu.edu/~213/codeStyle.html

C Exercise: `$ man 3 getopt`

- `int getopt(int argc, char * const argv[], const char *optstring);`
- `getopt` returns -1 when done parsing
- `optstring` is string with command line arguments
  - Characters followed by colon require arguments
    - Find argument text in `char *optarg`
  - `getopt` can't find argument or finds illegal argument sets `optarg` to "?"
  - Example: "`abc:d:`"
    - a and b are boolean arguments (not followed by text)
    - c and d are followed by text (found in `char *optarg`)

C Exercise: `getopt example`

- `$ make getopt`
- $ ./getopt_example -n 10
- $ ./getopt_example -n 10 -v
- Try switching the parameters around. Does it still work?
- Modify the example to include a step size
  - $ ./getopt_example -n 10 -s 2 -v should count 0 – 10 by 2s
  - The default step size should be 1

C Exercise: `getopt`

- `Now write your own!`
- Open a new file called getopt.c
- Write a simple hashing function with usage
  - -s : string with length at most 100
  - -i : max index
  - -v : optional verbose flag
  - -f : which hashing function to use
  - When f is off, sum all characters % i
  - When f is set, multiply each character by it’s index before adding

If you get stuck…

- C Bootcamp next Sunday 2/25
- Reread the writeup
- Look at CS:APP Chapter 6
- Review lecture notes (http://cs.cmu.edu/~213)
- Come to Office Hours (Sunday to Thursday, 5-9pm WH-5207)
- Post private question on Piazza
- `man malloc`, `man valgrind`, `man gdb`

Appendix

- Valgrind
- Clang / LLVM
- Cache Structure

Appendix: Valgrind

- Tool used for debugging memory use
  - Finds many potential memory leaks and double frees
  - Shows heap usage over time
  - Detects invalid memory reads and writes
  - To learn more… man valgrind

- Finding memory leaks
  - `$ valgrind –leak-resolution=high –leak-check=full –show-reachable=yes –track-fds=yes ./myProgram arg1 arg2`

Appendix: Clang / LLVM

- Clang is a (gcc equivalent) C compiler
  - Support for code analyses and transformation
  - Compiler will check you variable usage and declarations
  - Compiler will create code recording all memory accesses to a file
  - Useful for Cache Lab Part B (Matrix Transpose)

Appendix: Cache Structure