15-213 Recitation: C Review

TA’s
11 Feb 2019
Agenda

- Logistics
- C Assessment
- C Programming Style
- C Exercise
- Stack Frame
- Attack Lab Introduction
Logistics

- Bomb Lab is due Thursday at midnight!
  - “But if you wait until the last minute, it only takes a minute!” – NOT!
  - Don’t waste your grace days on this assignment!

- Attack Lab will be released shortly thereafter!
C Assessment

■ 3.5 Basic C Programming Questions

■ Take some time to write down your answer for each question
C Assessment: Question 1

- Which lines have a problem and how can you fix it?

```c
int main(int argc, char** argv) {
    int *a = (int*) malloc(213 * sizeof(int));
    for (int i=0; i<213; i++) {
        if (a[i] == 0) a[i]=i;
        else a[i]=-i;
    }
    return 0;
}
```
C Assessment: Question 1

- malloc can fail!

```c
int main(int argc, char** argv) {
    int *a = (int*) malloc(213 * sizeof(int));
    if (a == NULL) return 0;
    for (int i=0; i<213; i++) {
        if (a[i] == 0) a[i]=i;
        else a[i]=-i;
    }
    return 0;
}
```
C Assessment: Question 1

Allocated memory is not initialized!

```c
int main(int argc, char** argv) {
    int *a = (int*) calloc(213, sizeof(int));
    if (a == NULL) return 0;
    for (int i=0; i<213; i++) {
        if (a[i] == 0) a[i]=i;
        else a[i]=-i;
    }
    return 0;
}
```
C Assessment: Question 1

- Declaring variables inside a for loop requires `-std=c99`

```c
int main(int argc, char** argv) {
    int *a = (int*) calloc(213, sizeof(int));
    if (a == NULL) return 0;
    for (int i=0; i<213; i++) {
        if (a[i] == 0) a[i]=i;
        else a[i]=-i;
    }
    return 0;
}
```
C Assessment: Question 1

- All allocated memory must be freed!

```c
int main(int argc, char** argv) {
    int *a = (int*) calloc(213, sizeof(int));
    if (a == NULL) return 0;
    for (int i=0; i<213; i++) {
        if (a[i] == 0) a[i]=i;
        else a[i]=-i;
    }
    free(a);
    return 0;
}
```
C Assessment: Question 2

- What are the values of A and B?

```c
#define SUM(x, y) x + y

int sum(int x, int y) {
    return x + y;
}

int A = SUM(2, 1) * 3;
int B = sum(2, 1) * 3;
```
C Assessment: Question 2

- What is wrong with our macro SUM?

```c
#define SUM(x, y) x + y

int sum(int x, int y) {
    return x + y;
}

int A = SUM(2, 1) * 3; // A = 2 + 1 * 3 = 5!? 
int B = sum(2, 1) * 3;  // B = 9
```
C Assessment: Question 2

- Use parenthesis around result!

```c
#define SUM(x, y) (x + y)

int sum(int x, int y) {
    return x + y;
}

int A = SUM(2, 1) * 3; // A = 9
int B = sum(2, 1) * 3; // B = 9
```
C Assessment: Question 2 Part B

- What are the values of A and B?

```c
#define MULT(x, y) (x * y)

int mult(int x, int y) {
    return x * y;
}

int A = MULT(2, 0 + 1) * 3;
int B = mult(2, 0 + 1) * 3;
```
C Assessment: Question 2 Part B

- What is wrong with our macro `MULT`?

```c
#define MULT(x, y) (x * y)

int mult(int x, int y) {
    return x * y;
}

int A = MULT(2, 0 + 1) * 3; // A = (2 * 0 + 1) * 3 = 3?!
int B = mult(2, 0 + 1) * 3; // B = 6
```
C Assessment: Question 2 Part B

- Use parenthesis around macro arguments (and result)!

```c
#define MULT(x, y) ((x) * (y))

int mult(int x, int y) {
    return x * y;
}

int A = MULT(2, 0 + 1) * 3; // A = ((2) * (0 + 1)) * 3 = 6
int B = mult(2, 0 + 1) * 3; // B = 6
```
C Assessment: Question 2

• Macros are good for compile-time decisions
  • Assert, requires, etc
  • dbg_print

• Macros are not functions and should not be used interchangeably
What lines make safe_int_malloc not so safe?

```c
int *safe_int_malloc(int *pointer) {
    pointer = malloc(sizeof(int));
    if (pointer == NULL) exit(-1);
    return &pointer;
}
```
C Assessment: Question 3

- pointer is a local copy of the pointer!

```c
1 int *safe_int_malloc(int **pointer) {
2     *pointer = malloc(sizeof(int));
3     if (pointer == NULL) exit(-1);
4     return &pointer;
5 }
```
C Assessment: Question 3

■ &pointer is a location on the stack in safe_int_malloc’s frame!

```c
1 int **safe_int_malloc(int **pointer) {
2     *pointer = malloc(sizeof(int));
3     if (pointer == NULL) exit(-1);
4     return pointer;
5 }
```
C Assessment Conclusion

- Did you answer every question correctly? If not…
  - Attend the C Bootcamp on Feb 24

- Was the test so easy you were bored? If not…
  - Attend the C Bootcamp on Feb 24

- When in doubt…
  - Attend the C Bootcamp on Feb 24

- 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! 😎
  - Refer to full 213 Style Guide: http://cs.cmu.edu/~213/codeStyle.html
C Exercise

- Learn to use getopt
  - Extremely useful for Cache Lab
  - Processes command line arguments

- Let’s write a Pythagorean Triples Solver!
  - Pair up!
  - Login to a shark machine
  - $ wget http://cs.cmu.edu/~213/recitations/rec6.tar
  - $ tar xvf rec6.tar
  - $ cd rec6

- But first, a simple getopt example...
  - $ vim getopt-example.c
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: C Hints and Math Reminders

- \( a^2 + b^2 = c^2 \)
  - \( \Rightarrow a = \sqrt{c^2 - b^2} \)
  - \( \Rightarrow b = \sqrt{c^2 - a^2} \)
  - \( \Rightarrow c = \sqrt{a^2 + b^2} \)
  - \( \Rightarrow 3^2 + 4^2 = 5^2 \)

- String to float in C:
  
  ```c
  #include <stdlib.h>
  float atof(const char *str);
  ```

- Square root in C:
  
  ```c
  #include <math.h>
  float sqrt(float x);
  ```
x86-64/Linux Stack Frame

**Current Stack Frame (“Top” to Bottom)**
- “Argument build:”
  - Parameters for function about to call
- Local variables
  - If can’t keep in registers
- Saved register context
- Old frame pointer (optional)

**Caller Stack Frame**
- Return address
  - Pushed by call instruction
- Arguments for this call
Attack Lab

• We’re letting you hijack programs by running buffer overflow attacks on them.
  • Is that not justification enough?

• To understand stack discipline and stack frames

• To defeat relatively secure programs with return oriented programming