How to Write a .c File

15-213: Introduction to Computer Systems

Recitation 6, Oct 1, 2012

Alexander Malyshev (amalyshe)

Section A, 10:30a – 11:20p, WeH 4623

Agenda

- **Buffer overflow**
- Writing a C program
- Makefiles
- Revision Control

Buffer Overflow

- We have an IA32 stack frame with an array on it and nothing else and we call gets(buf)
  - `char buf[8];`
- Does anything bad happen when we type in:
  - “1234567”

Buffer Overflow

- We have an IA32 stack frame with an array on it and nothing else and we call gets(buf)
  - `char buf[8];`

- Does anything bad happen when we type in:
  - “1234567” // nothing bad happens
  - “12345678912”

Buffer Overflow

- We have an IA32 stack frame with an array on it and nothing else and we call gets(buf)
  - `char buf[8];`

- Does anything bad happen when we type in:
  - “1234567” // nothing bad happens
  - “12345678912” // we overwrite the old ebp with // 0x00323139
  - “123456789123456”

Buffer Overflow

- We have an IA32 stack frame with an array on it and nothing else and we call gets(buf)
  - `char buf[8];`

- Does anything bad happen when we type in:
  - “1234567” // nothing bad happens
  - “12345678912” // we overwrite the old ebp with
    // 0x00323139
  - “123456789123456” // old ebp = 0x33323139
    // return address = 0x00363534

Buffer Overflow True/False

- A buffer overflow attack can only be executed on programs that use the gets() function

Buffer Overflow True/False

- A buffer overflow attack can only be executed on programs that use the gets() function
  - False, you don’t need gets() to write past the length of a buffer

Buffer Overflow True/False

- A buffer overflow attack can only be executed on programs that use the gets() function
  - False, you don’t need gets() to write past the length of a buffer
- Buffer overflow attacks all occur on the stack

Buffer Overflow True/False

- A buffer overflow attack can only be executed on programs that use the gets() function
  - False, you don't need gets() to write past the length of a buffer
- Buffer overflow attacks all occur on the stack
  - False, a buffer can be allocated on the heap and someone can just as easily write past the end of it, but they won't be attacking the return address directly anymore

Agenda

- Buffer overflow
- **Writing a C program**
- Makefiles
- Revision Control

Writing Code from Scratch

- We want to write a C program that takes the length and width of a rectangle, and prints its area

- The length will be specified by the “-x” flag, and the width will be specified by the “-y” flag
  - `./area -x 5 -y 7`

- Any bad arguments for -x and -y should cause the program to print 0

Writing Code from Scratch

```
$ cat area.c
int main(int argc, char **argv) {
      int x, y;

      // “somehow” get arguments into x and y

      printf(“Area: %d\n”, x * y);
      return 0;
}
```

Writing Code from Scratch

- We could iterate through argv and do a strcmp with “-x” and “-y” to find our integers but that can quickly get messy
  - Doesn’t scale well for many arguments

- Use the getopt() function instead

Writing Code from Scratch

- getopt() takes argc, and argv and a format string, and then returns the type of the current argument and moves the value of the current argument into a global variable named “optarg”

- Uses less code, and can handle all sorts of complicated arguments

Agenda

- Buffer overflow
- Writing a C program
- **Makefiles**
- Revision Control

Makefiles

- A way of building multiple source files into an executable
- Old, crufty, and the syntax isn't pretty
- We don't recommend writing one from scratch, most people just copy existing ones they find online
  - You'll get one for cachelab and all subsequent labs

Makefiles

- You might want to modify the starter Makefile for future labs
  - Such as moving some of your code into separate .c files
- Should know how to modify/add make rules
- If you don't remember what all the weird variables are ($?, $@, etc), Google is your friend
  - Don't bother memorizing them

Makefiles

```
CC = gcc
CFLAGS = -Wall –Wextra –g

all: helloworld

helloworld: helloworld.o
	# this HAS to be a hard tab, using spaces will not work
	$(CC) $(CFLAGS) $(LDFLAGS) –o helloworld helloworld.o

helloworld.o:
	$(CC) $(CFLAGS) -c helloworld.c

clean:
	rm –f helloworld helloworld.o

.PHONY: clean
```

Makefiles

```
CC = gcc
CFLAGS = -Wall –Wextra –g

all: helloworld

helloworld: helloworld.o
	# this HAS to be a hard tab, using spaces will not work
	$(CC) $(CFLAGS) $(LDFLAGS) –o $@ $^

helloworld.o:
	$(CC) $(CFLAGS) -c $<

clean:
	rm –f helloworld helloworld.o

.PHONY: clean
```

Makefiles

```
CC = gcc
CFLAGS = -Wall –Wextra –g

all: helloworld

helloworld: helloworld.o
	# this HAS to be a hard tab, using spaces will not work
	$(CC) $(CFLAGS) $(LDFLAGS) –o $@ $^

%.o: %.c
	$(CC) $(CFLAGS) –c $<

clean:
	rm –f helloworld *.o

.PHONY: clean
```

Agenda

- Buffer overflow
- Writing a C program
- Makefiles
- **Revision Control**

Revision Control

- A set of tools to help keep track of multiple versions of a project
  - Most commonly used to manage source code
- Want to keep history of your changes
  - **Who** changed **what** and **when**
- This will be super useful when you work with more than one person (proxylab)

Revision Control

- Many programs exist for this purpose
  - CVS, Subversion (svn), darcs, git
- We recommend that you use git
  - Shout out to stuco 98-174

git

- http://git-scm.com/book is your best resource for learning git
  - Extremely helpful, and the initial chapters get you started very quickly

- You'll really only need a few commands when working by yourself
  - git {init, add, commit, log, status}
  - If you are willing, read up on branching, it will be super useful for malloclab

Summary

- Buffer overflow
- Writing a small C program from scratch
- Makefiles and their quirks
- Revision Control (aka. Please use git)