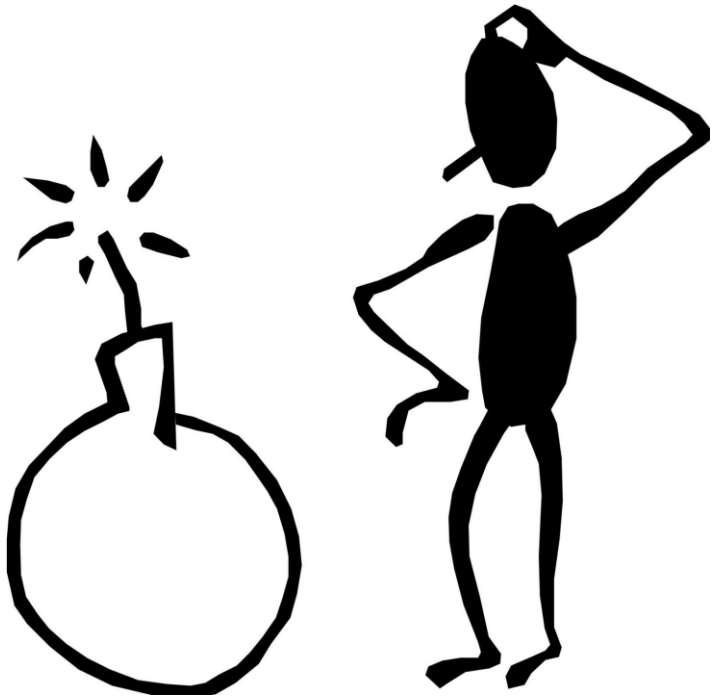
15-213 Recitation 4: Bomb Lab

19 Sept 2016

Agenda

- Bomb Lab Overview
- Introduction to GDB
- Bomb Lab Practice

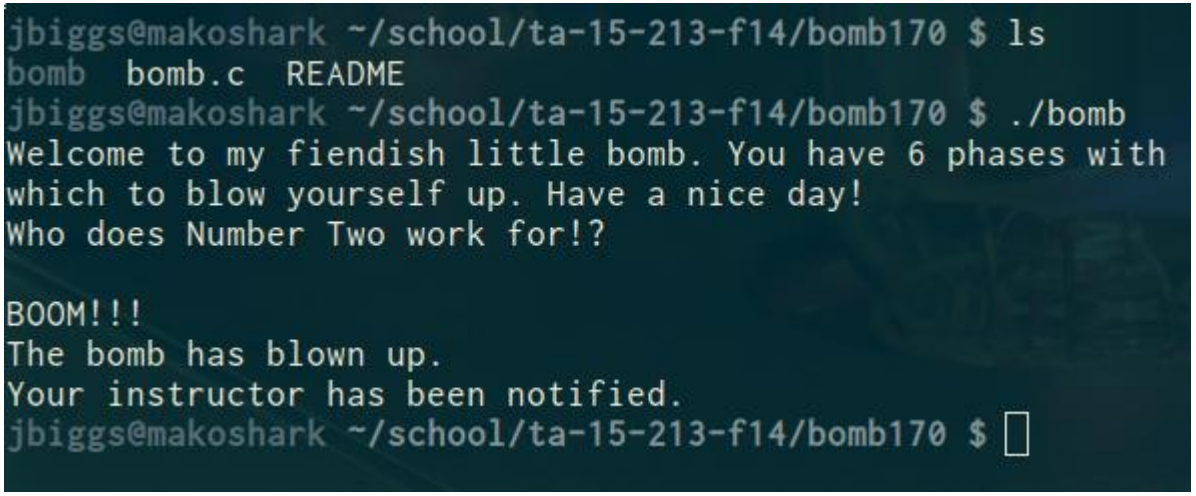


Downloading Your Bomb

- **Please read the writeup.** ***Please read the writeup.*** ***Please read the writeup.***
- Your bomb is **unique** to you. Dr. Evil has created one ~~million~~ billion bombs, and can distribute as many new ones as he pleases.
- if you download a second bomb, it will be different from the first!
- Bombs have six phases which get progressively ~~harder~~ more fun to use.
- Bombs can only run on the shark clusters. They **will** blow up if you attempt to run them locally.

Exploding Your Bomb

- Blowing up your bomb notifies Autolab.
- Dr. Evil takes **0.5** of your points each time the bomb explodes.
- Inputting the correct string moves you to the next phase.
- Jumping between phases detonates the bomb – you have to solve them in the given order.



Gdb

- Commandline debugger
- Learn to use it now

- The following slides will guide us through basic commands

We will use this notation:

- $ ls //This is a comment, $ indicates the commandline
- (gdb) // This indicates the command should be typed in gdb

Form pairs

- One student needs a laptop
- Login to a shark machine

$ wget http://www.cs.cmu.edu/~213/activities/rec4.tar

$ tar xf rec4.tar

$ cd rec4

$ cat act1.c // What does the act1.c do?

$ make

$ gdb act1

Activity 1

(gdb) disassemble main

(gdb) break (put call name here)

(gdb) run

Q. Does the program run to completion? If not, where does it stop?

(gdb) continue

Q. What does the program /debugger do after continuing?

(gdb) disable 1

(gdb) run 15213

Q. What happens now?

Activity 1 cont.

(gdb) disassemble main

(gdb) print (char*) 0x … // Find a $0x… in the assembly

Q. Does the printed value correspond to anything in the source code?

(gdb) break main

(gdb) run

(gdb) print argv[0]

(gdb) quit

Activity 2

$ gdb act2

(gdb) break main

(gdb) run

(gdb) print /x $rsi

(gdb) print /x $rdi

Q. RSI and RDI are registers that pass the first two arguments. Looking at their values, which is the first argument to main? Why?

(gdb) disassemble main

(gdb) break (what main calls)

(gdb) continue

Q. How could you view the values that have been passed to stc?

Activity 2 cont.

(gdb) run 18213 // gdb will ask if you want to start from the beginning, yes

Q. Does the program run to completion?

[continue execution until stc is called]

(gdb) disassemble // Disassemble the current function

(gdb) nexti // Run 1 instruction

(gdb) “\n” // Just press enter

(gdb) “\n”

Q. Which function is execution in now?

(gdb) disassemble

Q. Where are the “=>” characters printed on the left side?

Q. Print the values of the three registers used in this function. Which register(s) have values that you expect?

(gdb) quit // gdb will ask if you want to kill the process, yes.

Activity 3

// act3 expects two numbers as commandline arguments

// These numbers must cause compare to return 1

$ cat act3.c

$ gdb act3

Q. Which register holds the return value from a function? (Hint: the register is used in the instruction after main calls compare)

(gdb) disassemble compare

Q. Where is the return value set in compare?

(gdb) break compare

// Run act3 with two numbers

Q. Using nexti, how does the value in register rbx change leading to the cmp instruction?

```
// reg += val
Add <val>, %reg
```

```
// reg - val
Cmp <val>, %reg
// if (cc) reg = 1
Set<cc> %reg
```

Activity 4

// Use what you have learned to get act4 to

// print “Finish” in the function compute(int)

// The source code and TA are available if you get stuck

$ gdb act4

```
// Jump table at addr
// (gdb) x /5gx <addr>
Jmpq *<addr>(%reg)

// reg >>= val
Shl <val>, %reg

// reg - val
Cmp <val>, %reg
// reg & val
Test <val>, %reg
// jump via the condition codes
J<cc> <addr>
```

```
// No operation
Nop …

// reg &= val
And <val>, %reg

// reg >>= val
Sar <val>, %reg

// reg += val
Add <val>, %reg

// reg -= val
Sub <val>, %reg
```