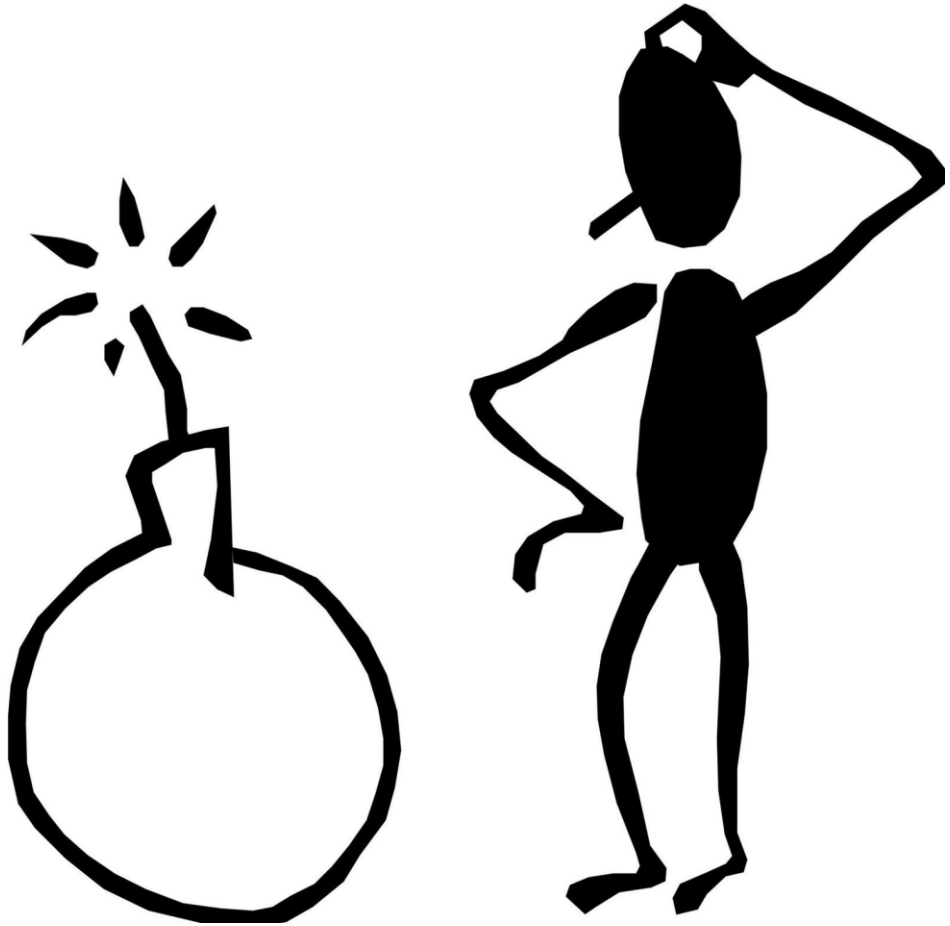
Introduction to Computer Systems
Bomb Lab

2022 Spring, CSE3030

Sogang University

Agenda

- Bomb Lab Overview
- Assembly Refresher
- Introduction to GDB
- Unix Refresher
- Bomb Lab Demo



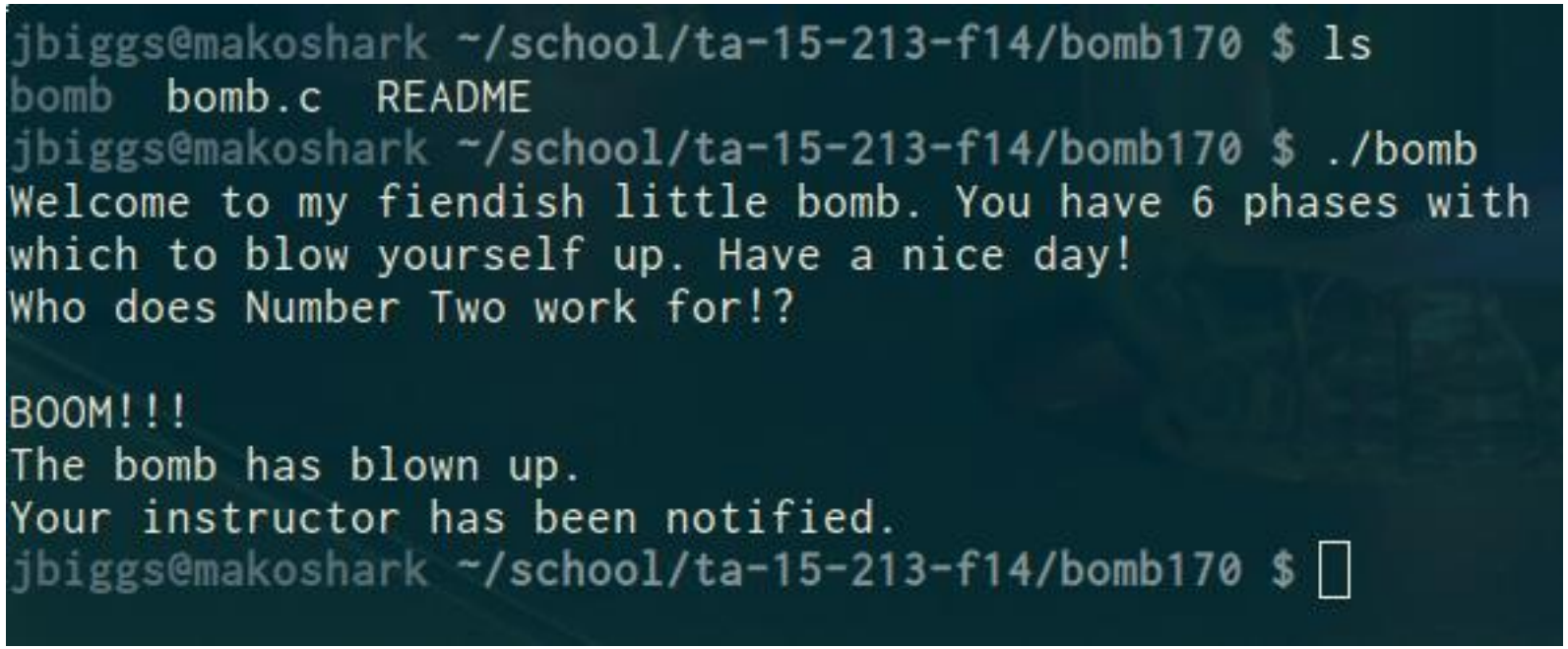
Downloading Your Bomb

- Run the following code on the cspro server.
  - wget http://csapp.cs.cmu.edu/3e/bomb.tar
  - tar xvf bomb.tar

- Bombs have six phases which get progressively ~~harder~~ more fun to use.

Exploding Your Bomb

- Inputting the right string moves you to the next phase.
  - Jumping between phases detonates the bomb



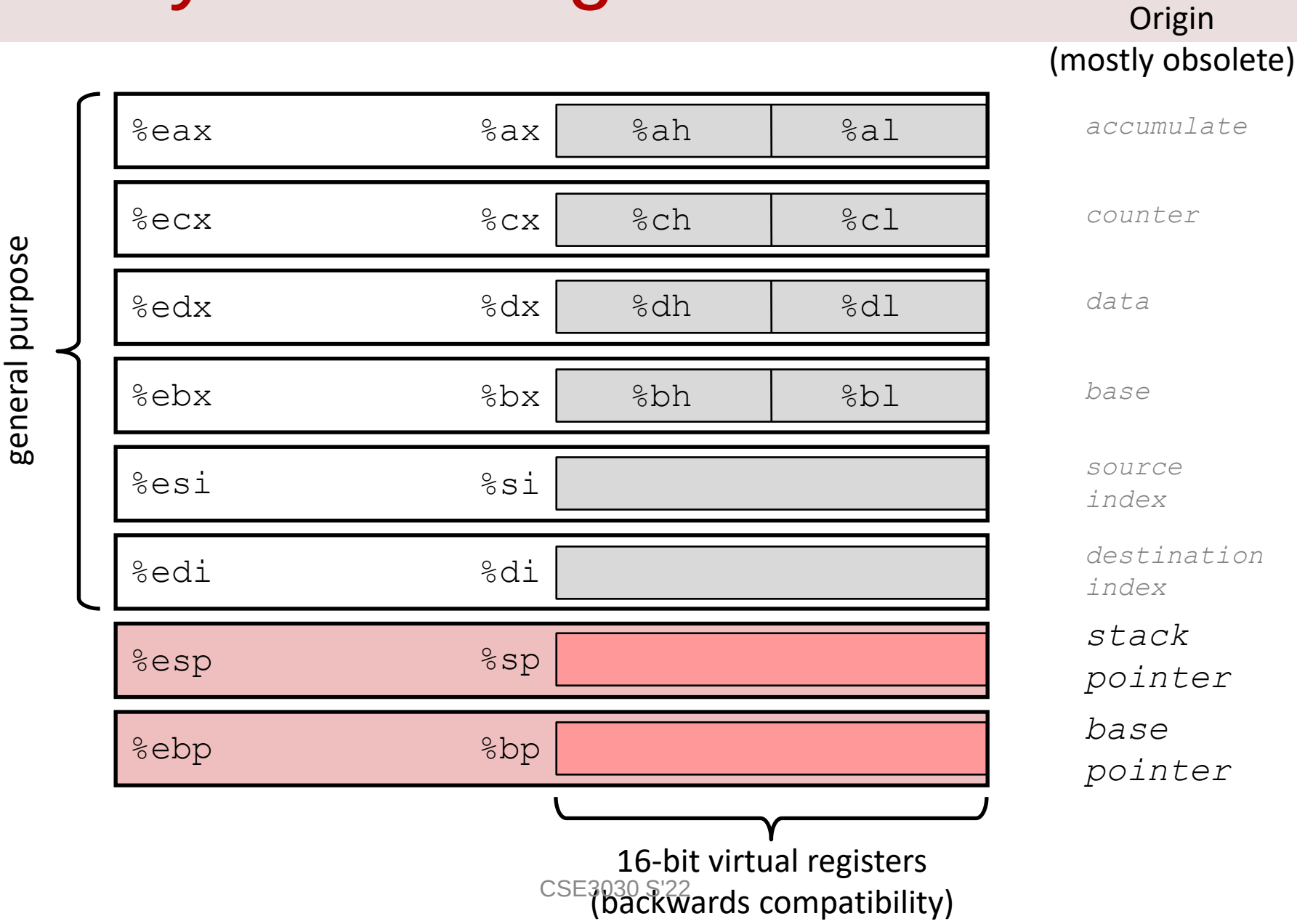
Examining Your Bomb

- You get:
  - An executable
  - A readme
  - A heavily redacted source file
- Source file just makes fun of you.
- Outsmart Dr. Evil by examining the executable

x64 Assembly: Registers

Return	%rax	%eax	%r8	%r8d	Arg 5
	%rbx	%ebx	%r9	%r9d	Arg 6
Arg 4	%rcx	%ecx	%r10	%r10d	
Arg 3	%rdx	%edx	%r11	%r11d	
Arg 2	%rsi	%esi	%r12	%r12d	
Arg 1	%rdi	%edi	%r13	%r13d	
Stack ptr	%rsp	%esp	%r14	%r14d	
	%rbp	%ebp	%r15	%r15d	

Some History: IA32 Registers



x64 Assembly: Operands

Type	Syntax	Example	Notes
Constants	Start with $	`$-42` `$0x15213b`	Don't mix up decimal and hex
Registers	Start with %	`%esi` `%rax`	Can store values or addresses

Memory Locations	Parentheses around a register or an addressing mode	`(%rbx)`
		`0x1c(%rax)`
		`0x4(%rcx, %rdi, 0x1)`
			Parentheses dereference. Look up addressing modes!

x64 Assembly: Arithmetic Operations

```
        mov %rbx, %rdx
    add (%rdx), %r8
        mul $3, %r8
        sub $1, %r8
lea (%rdx,%rbx,2), %rdx
```

- **Effect**

rdx = rbx
r8 += value at rdx
r8 *= 3
r8--
rdx = rdx + rbx*2

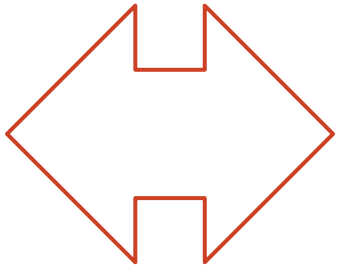
  - *Doesn't dereference*

- **Instruction**

x64 Assembly: Comparisons

- Comparison, `cmp`, compares two values
  - Result determines next conditional jump instruction
- `cmp b,a` computes `a-b`, `test b,a` computes `a&b`
- Pay attention to **operand order**

```
cmpl %r9, %r10
jg 8675309
```



If `%r10 > %r9`, then jump to `8675309`

x64 Assembly: Jumps

Instruction	Effect	Instruction	Effect
`jmp`	Always jump	`ja`	Jump if above (unsigned >)
`je/jz`	Jump if eq / zero	`jae`	Jump if above / equal
`jne/jnz`	Jump if !eq / !zero	`jb`	Jump if below (unsigned <)
`jg`	Jump if greater	`jbe`	Jump if below / equal
`jge`	Jump if greater / eq	`js`	Jump if sign bit is 1 (neg)
`jl`	Jump if less	`jns`	Jump if sign bit is 0 (pos)
`jle`	Jump if less / eq		

x64 Assembly: A Quick Drill

```
cmp $0x15213, %r12
jge deadbeef

cmp %rax, %rdi
jae 15213b

test %r8, %r8
jnz (%rsi)
```

If _____, jump to add
r 0xdeadbeef

If _____, jump to add
r 0x15213b

If _____, jump to
.

x64 Assembly: A Quick Drill

```
cmp $0x15213, %r12
jge deadbeef

cmp %rax, %rdi
jae 15213b

test %r8, %r8
jnz (%rsi)
```

If `%r12 >= 0x15213`, ju
mp to `0xdeadbeef`

x64 Assembly: A Quick Drill

```
cmp $0x15213, %r12
jge deadbeef

cmp %rax, %rdi
jae 15213b

test %r8, %r8
jnz (%rsi)
```

If the unsigned value of `%rdi` is at or above the unsigned value of `%rax`, j ump to `0x15213b`.

x64 Assembly: A Quick Drill

```
cmp $0x15213, %r12
jge deadbeef

cmp %rax, %rdi
jae 15213b

test %r8, %r8
jnz (%rsi)
```

If `%r8 & %r8` is not zero, jump to the address stored in `%rsi`.

Diffusing Your Bomb

- `objdump -t bomb` examines the symbol table
- `objdump -d bomb` disassembles all bomb code
- `strings bomb` prints all printable strings
- **`gdb bomb`** will open up the **G**NU **Deb**ugger
  - Examine while stepping through your program
    - registers
    - the stack
    - contents of program memory
    - instruction stream

Using `gdb`

- `break <location>, b <location>`
  - Stop execution at function name or address
  - Reset breakpoints when restarting `gdb`
  - `break phase_1, break *phase_1+32`
- `run <args>, r`
  - Run program with args `<args>`
  - Convenient for specifying text file with answers
- `disas <fun>`, but **not** `dis`
- `continue, c`
  - Run program until the next breakpoint
- `stepi / nexti`
  - Steps / does not step through function calls
- `delete`
  - delete all breakpoints
- `clear <location>`
  - Clean break points at function name or address

Using gdb

- `info registers, info r`
  - Print hex values in every register
- `print` (`/x` or `/d`) `$eax` - Yes, use `$`
  - Print hex or decimal contents of `%eax`
- `x $register, x 0xaddress`
  - Prints what's in the register / at the given address
  - By default, prints one word (4 bytes)
  - Specify format: /s, /[num][size][format]
    - `x/8a 0x15213`
    - `x/4wd 0xdeadbeef`

`sscanf`

- Bomb uses `sscanf` for reading strings
- Figure out what phase expects for input
- Check out `man sscanf` for formatting string details

If you get stuck

- **Please read the writeup. *Please read the writeup*. *Please Read The Writeup.***
  - ***http://csapp.cs.cmu.edu/3e/bomblab.pdf***
- CS:APP Chapter 3
- View lecture notes and course FAQ at CyberCampus.
- `man gdb, man sscanf, man objdump`