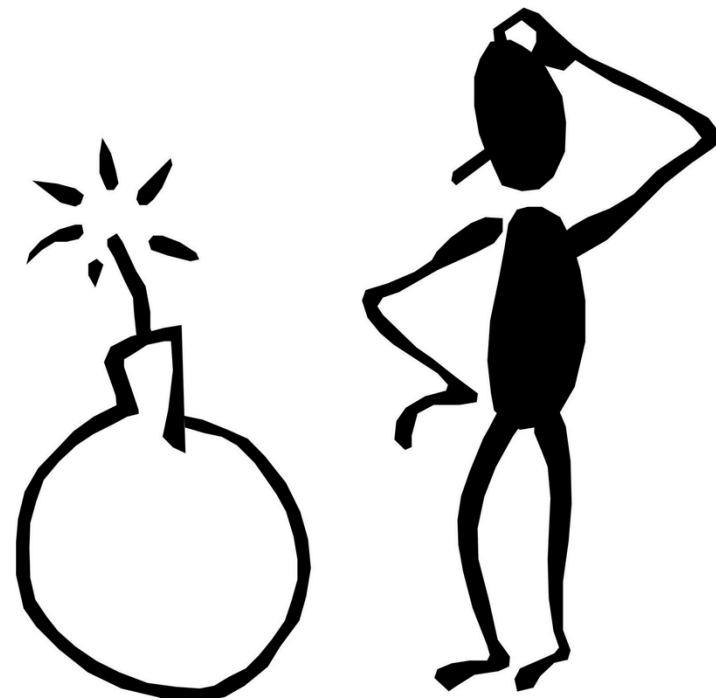


Bomb Lab

cs304

Agenda

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



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.
- Bombs have six phases which get progressively ~~harder~~ more fun to use.
- Bombs can only run on the machines in 121. 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.
- Inputting the right string moves you to the next phase.

```
jujbiggs@makoshark ~/school/ta-15-213-f14/bomb170 $ ls
bomb  bomb.c  README
jujbiggs@makoshark ~/school/ta-15-213-f14/bomb170 $ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Who does Number Two work for!?
```

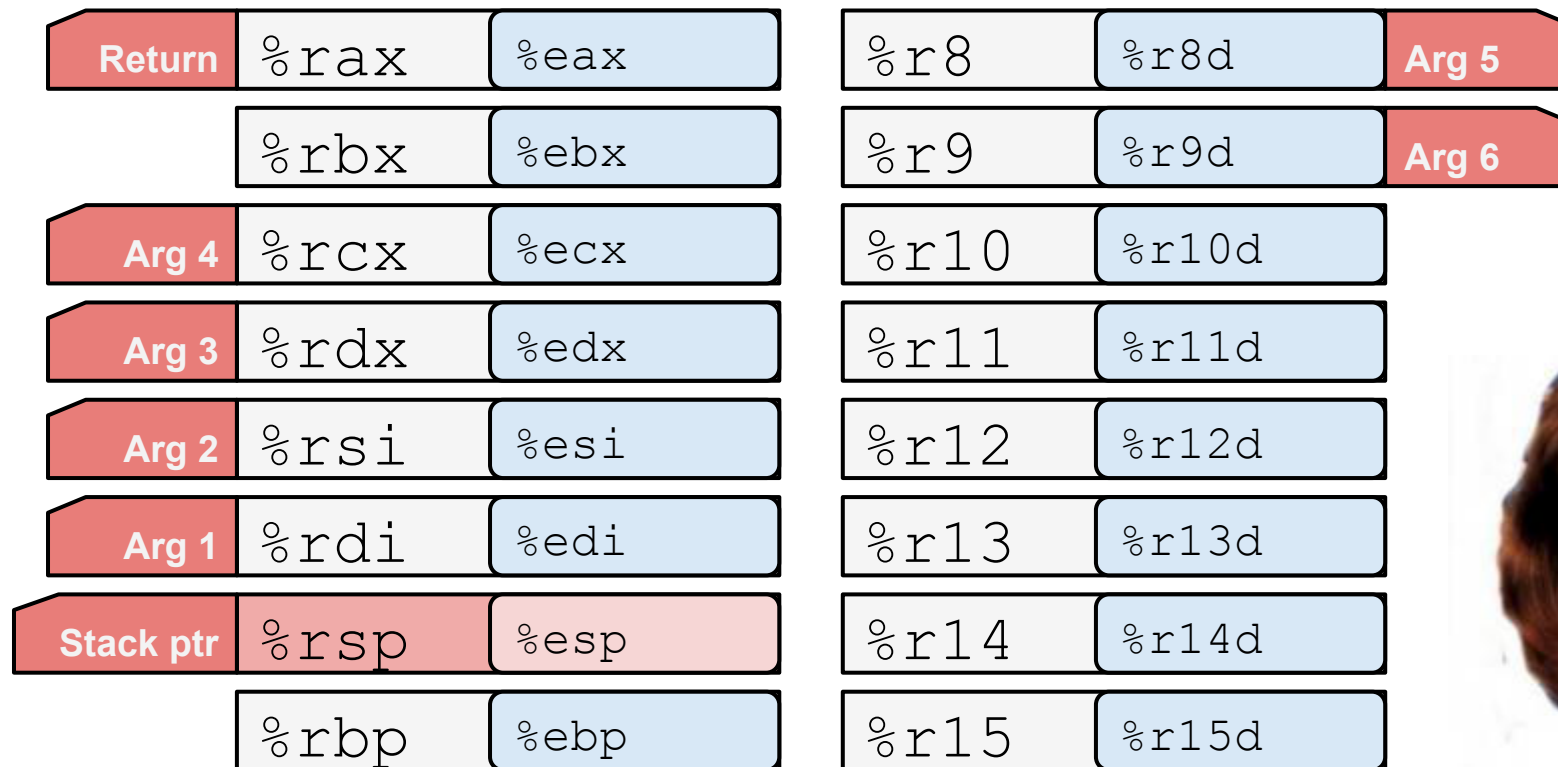
BOOM!!!
The bomb has blown up.
Your instructor has been notified.
jujbiggs@makoshark ~/school/ta-15-213-f14/bomb170 \$ □

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



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

Instruction	Effect
<code>mov %rbx, %rdx</code>	<code>rdx = rbx</code>
<code>add (%rdx), %r8</code>	<code>r8 += value at rdx</code>
<code>mul \$3, %r8</code>	<code>r8 *= 3</code>
<code>sub \$1, %r8</code>	<code>r8--</code>
<code>lea (%rdx,%rbx,2), %rdx</code>	<code>rdx = rdx + rbx*2</code> <ul style="list-style-type: none">■ <i>Doesn't dereference</i>

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

x64 Assembly: A Quick Drill

```
cmp $0x15213, %r12    If _____, jump to addr  
jge deadbeef          0xdeadbeef
```

```
cmp %rax, %rdi        If _____, jump to addr  
jae 15213b            0x15213b
```

```
test %r8, %r8        If _____, jump to _____.  
jnz (%rsi)
```

x64 Assembly: A Quick Drill

```
cmp $0x15213, %r12    if %r12 >= 0x15213,  
jge deadbeef         jump to 0xdeadbeef
```

```
cmp %rax, %rdi  
jae 15213b
```

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

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`, jump 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 **GNU Debugger**
 - Examine while stepping through your program
 - registers
 - the stack
 - contents of program memory
 - instruction stream

Using gdb

- `break <location>`
 - Stop execution at function name or address
 - Reset breakpoints when restarting `gdb`
- `run <args>`
 - Run program with `args <args>`
 - Convenient for specifying text file with answers
- `disas <fun>`, **but not** `dis`
- `stepi / nexti`
 - `Steps /` does not step through function calls

Using gdb

- `info registers`
 - 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 0xdeadbef`

`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.
- CS:APP Chapter 3
- `man gdb`, `man sscanf`, `man objdump`

Unix Refresher

You should know `cd`, `ls`, `scp`, `ssh`, `tar`, and `chmod` by now. Use `man <command>` for help.
`<Control-C>` exits your current program.

