## 15-213 Recitation Bomblab

Your TAs Friday, January 24th

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

1

## Reminders

- datalab is due on *Tuesday (Jan 28)*.
- **bomblab** is out! Due *February 6th*.
- **Cprogramming lab** was due last day to submit is today!
- Bootcamp 2: *Debugging & GDB* is pre-recorded. Watch Ed for the link.

## Agenda

- Assembly Refresher
- Preview: Calling Conventions
- Intro to bomblab
- bomblab defuse kit
- gdb activity

## **Assembly Refresher**

## **Reading Assembly**

We will use AT&T syntax in this class:

movq	Src,	Dest	movq	Dest,	Src
addq	Src,	Dest	addq	Dest,	Src

### AT&T

### Intel

If you get stuck, refer to our assembly cheat sheet!

$ \begin{array}{cccc} \operatorname{corr} \operatorname{Src} \operatorname{Dart} & \operatorname{Deral} & \operatorname{zero} & \operatorname{corr} \operatorname{Src} & \operatorname{Dart} & \operatorname{Dart} \operatorname{Dart} & Da$	Instructions		Arithmetic of	perations	Inst	ruction suffixe
$ \begin{array}{llllllllllllllllllllllllllllllllllll$	Data movemer	ıt				
	nova Src. Dest	Dest = Src	deca Dest	Dest = Dest - 1		
avzká Src. Det Sver Sv. Det Sver Sver Sver Sv. Det Sver Sver Sver Sver Sver Sver Sver Sver		Dest (quad) = Src (byte), sign-extend		Dest = Dest + Src		
	novzba Src.Dest	Dest (ouad) = Src (byte), zero-extend		Dest = Dest - Src	d d	uad (8 bytes)
$ \begin{array}{c c c c c c c c c c c c c c c c c c c $				Dest = Dest * Src		
$ \begin{array}{cccc} \operatorname{score} \operatorname{Str. Det} & \operatorname{Deal} & \operatorname{Jray} & \operatorname{erq}\operatorname{Str. Det} & \operatorname{Deal} & \operatorname{Deal} & \operatorname{Str. } & \operatorname{Str. } & \operatorname{Deal} & \operatorname{Str. } & $	Conditional m	01/0		Dest = Dest ^ Src	Con	dition codes
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$				Dest = Dest   Src	an	0.00
absort Str., Date         New option         New option         SP Sign Parg           accord Str., Date         New option         Set of Sign Parg         SP Sign Parg           accord Str., Date         Creater of sign (digod 2)         and k be         Date = - Det         SP Sign Parg           accord Str., Date         Creater of sign (digod 2)         and k be         Date = - Det         SP Sign Parg           accord Str., Date         Creater of sign (digod 2)         and k be         Date = - Det         Date = - Det         SP Sign Parg           accord Str., Date         Creater of sign (digod 2)         and k be         Date = - Det         Date = - Det         Med (mathed)         Integer register           accord Str., Date         Lano (mathed)         Addressing modes         Met         Net						
Carbor Sac, Indi.     Avgatus:     note to be the set of the set				Dest = - Dest		
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$			note Dest	$Dest = \sim Dest$		
$ \begin{array}{llllllllllllllllllllllllllllllllllll$					OF	Overnow r lag
cov/S Sr., Det     Loss (agaod <)				$Dest = Dest \gg k$ (arithmetic)		
aboy 1.8     Low (unique C)     Addressing modes     %aa       acova Sa, Dab     Abow (unique C)     Status 1.8     %aa       acova Sa, Dab     Abow (unique C)     Status 1.8     %aa       acova Sa, Dab     Abow (unique C)     Status 1.8     %aa       acova Sa, Dab     Below (unique C)     Status 1.8     %aa       acova Sa, Dab     Below (unique C)     Status 1.8     %aa       acova Sa, Dab     Below (unique C)     Status 1.8     %aa       acova Sa, Dab     Below (unique C)     Status 1.8     %aa       Control transfer     max     %ab     ala segments       caps Sci. Sci. Sci. Sci. Sci. Sci. Sci.     Normal     %ps     %ab apparents       ja 1abal     jump outor-segitor     Normal     %ps     %ab apparents       ja 1abal     jump outor-segitor     Diphoemeret     %11     Scatak register       ja 1abal     jump outor-segitor     Diphoemeret     %12     Calae saved       ja 1abal     jump jone (que c)     b)     b)     Calae saved       ja 1abal     jump jone (que c)     b)     b)     Calae saved       ja 1abal     jump jone (que c)     b)     b)     Calae saved       ja 1abal     jump jone (que c)     b)     b) <t< td=""><td></td><td></td><td>ahra k. Dest</td><td><math>Dest = Dest \gg k</math> (logical)</td><td>Inte</td><td>ger registers</td></t<>			ahra k. Dest	$Dest = Dest \gg k$ (logical)	Inte	ger registers
covo Ste, Det Abeve (rengial miniginal ≥) covor Ste, Det Beleve (range (miniginal ≥) site constant lategre value server Str. Asset Ste COS Stel Stel pi label jump neticular je label jump praster (raise je label jump praster (raise - ) je label jump kong (raigde -) je label jump kong (raigde -) praster - ) set Autor (raise - ) set Auto						
and the set of the se			Addamatica			
Cardy Step, Dat.         Below (ungiged          Immediate         Write 3id organization scores Step, Dat.           Control transfer         Sta V21			Addressing	modes		
Larrison Single Construction         State Margination         State Margination           Construction         Below or equal (unique)          State Margination         State Margination           Construction         State Construction         State Construction         State Construction         State Construction           Construction         State Constanet Construction         State Constanet Co			Terrar distant			
carbor set, toke         solar directory value         5, col         strangeneric           control transfer         song fr. first         %ff         different set           copp Sec.2, Stol.         set. COS Secl. Sec2         Normal         %ff         Calles aved           jsp 1abal         jump control         may control sec.0         %ff         Attach pointer           jsp 1abal         jump control         sec.0         %ff         Attach pointer           jsp 1abal         jump control         sec.0         %ff         Attach pointer           jsp 1abal         jump pointerregrad         %ff         Normal         %ff         Sec.0           jsp 1abal         jump pointerregrad         %ff         Normal         %ff         Sec.0         S						
Control transfer         sort $0^{-7}$ , $1^{-8}$ , $1^{-6}$ $0^{+6}$ (the avecl orgs fac, 2st.) $0^{+6}$ (as COS Stel. St.         Stel. $0^{-6}$ (the avecl $0^{-6}$ (ble ave	cmovbe Src, Dest	Below or equal (unsigned $\leq$ )		the second se		
$ \begin{array}{c c} \text{Outron transier} & & Source State $						
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	Control transf	er	movy or, A	Lax		
textq Stor. 2, stor. 3, stor. 2, stor. 4, stor.         [1], no starting length         %90         thh argument           parabal, jump parater (agend 2), stor.         stor. 1, fract         %10         Stor. Arguing fraction memory address         %10         Stor. Arguing fraction           parabal, jump parater (agend 2), stor.         bit parabal         jump parater (agend 2), bit parameters         %11         Stor. Arguing fraction         %12         Calles assed           parabal, jump parater (agend 2), jump parater (agend 2), jump parater (agend 2), bit paraginary parater (agend 2), bit parater (agend 2	cmpg Src2, Src1	Sets CCs Src1 Src2	• Normal			
$ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	testa Src2, Src1	Sets CCs Src1 & Src2	(R) Mem[Re	g[R]]		
js         js<	imp label	jump	R: register F	specifies memory address		
jns         label         jump not equal         Duplement         %(1)         Scatch register           jns         jabel         jump not equal         Diplement         %(2)         Calles and           jns         jabel         jump not equal         Diplement         %(2)         Calles and           jns         jabel         jump roter regular         %(1)         Calles and         %(1)         Calles and           js         jabel         jump poster (equal 5)         Do constant diplement         %(1)         Calles and           js         jabel         jump poster (equal 5)         Do constant diplement         %(1)         Calles and           js         jabel         jump poster (equal 5)         Do constant diplement         %(1)         Calles and           js         jabel         jump poster (equal 5)         Do constant diplement         %(1)         Calles and           js         jabel         jump poster (equal 5)         Do (hall 5)         Do (hall 5)         %(1)           js         jabel         jump poster (equal 6)         S(2)         Do (hall 5)         M(2)         S(2)           js         jabel         jump poster (equal 6)         S(2)         Do (hall 5)         Do (hall 5)         Do	ie label	jump equal	movq (%rcx	), %rax		
js						
jas label jump non-negative jas label jump non-negative jas label jump poster (agind $\supset$ ) jos label jump poster or equal (agind $\supseteq$ ) ios carstar (aligneternered D specifies offset %cf1 Calles awed ios carstar (aligneternered D specifies offset %cf2 Calles awed ios carstar (aligneter specifies offset %cf2 Calles awed ios carstar (aligneternered D specifies offset %cf2 Calles awed ios in aligneternered D specifies offset %cf2 Calles awed ios carstar (aligneternered D specifies %cf2 Calles awed ios carsta	is label	jump negative				
jg 1abal         jump greater (dgand 3)           ja 1abal         jump mode (exc oright)           ja 1abal         jump loss (exc oright)	ins label	jump non-negative				
$ \begin{array}{llllllllllllllllllllllllllllllllllll$	ig label	jump greater (signed >)	R: register s	pecifies start of memory region		
ji Labei jump loso cequi (ágned ≤) ja Labei jump loso cequi (ágned ≤) ja Labei jump loso (uságned >) D(Rh,RkS) Mom(Reg(Rb)+S*Rog(R)+D) D(rabei Jump losov (uságned >) D(rabei Jump losov	ige label	jump greater or equal (signed >)			%r15	Callee saved
ja labal jump above (randgrad >) D(R,K,S), Mum(Reg(R),4),+S*Reg(R),+D) ja labal jump below (randgrad <) D: constant displacement 1, 2, or 4 bytos pathą Src /krap = Krap 8, Mum(Prap) = Src Rt: base register: any or 60 starteger registers base - Mum(Prap) = Src Rt: base register: any or 66 starteger registers	jl label	jump less (signed $<$ )	movq 8(%rd	i), %rdx		
ja labal jump above (randgrad >) D(R,K,S), Mum(Reg(R),4),+S*Reg(R),+D) ja labal jump below (randgrad <) D: constant displacement 1, 2, or 4 bytos pathą Src /krap = Krap 8, Mum(Prap) = Src Rt: base register: any or 60 starteger registers base - Mum(Prap) = Src Rt: base register: any or 66 starteger registers	ile label	jump less or equal (signed $\leq$ )	. Indexed			
jb label jump below (unsigned <) D; constant displacement 1, 2, or 4 bytes punds Src %rsp = %rsp 8, Memi/Krsp] = Src Rb: have register: any of 8 integer registers popq Dest = Memi/Krsp], %rsp = %rsp + 8 Ri: index register: any exceent %eap	ja label			Mem[Reg[Rb]+S*Reg[Ri]+D]		
pushq Src %rsp = %rsp 8, Mem[%rsp] = Src Rb: base register: any of 8 integer registers popq Dest Dest = Mem[%rsp], %rsp = %rsp + 8 Ri: index register: any, excent %esp						
popq Dest Dest = Mem[%rsp], %rsp = %rsp + 8 Ri index register; any except %esp	pushq Src	%rsp = %rsp 8, Mem[%rsp] = Src				
		Dest = Mem[%rsp], %rsp = %rsp + 8				
call label push address of next instruction, jmp label S: scale: 1, 2, 4, or 8	call label					

#### x86-64 Reference Sheet (GNU assembler format)

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

## **Reading Assembly: Operands**

### Constants ("Immediate" Values)

Start with \$

**\$**-15213 *Decimal*  \$0x3b6d *Hex* 

### Registers

- Can store values or addresses
- Start with %

%rax "Return" Register %eax Low 32 bits of %rax

### **Memory Locations**

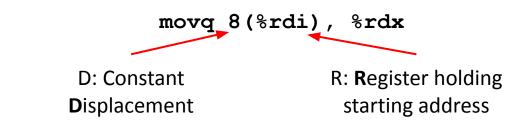
Parentheses around a register, or an addressing mode

(%rbx) Normal 0x1c(%rax) Displacement 

## **Reading Assembly: Addressing Modes**

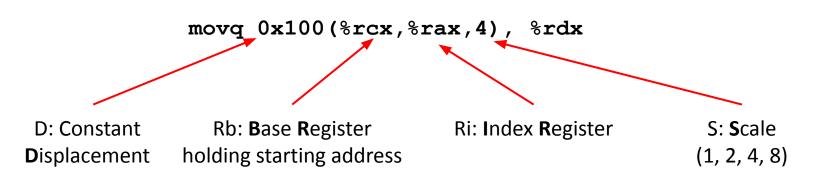
### Displacement

D(R) Mem[Reg[R] + D]



### Indexed

D(Rb, Ri, S) Mem[Reg[Rb] + S\*Reg[Ri] + D]



## **Reading Assembly: Examples**

Instruction	Effect
mov %rbx, %rdx	rdx = rbx
add (%rdx), %r8	r8 += value at address in rdx
mul \$3, %r8	r8 *= 3
sub \$1, %r8	r8
lea (%rdx, %rbx, 2), %rdx	rdx = rdx + rbx * 2
Ν	o dereferencing!

## **Reading Assembly: Comparisons**

### Example

- cmpl %r9, %r10 jg 8675309
- "If the value of one register is greater than the value in the other, then jump to 8675309"
- But which way around is it?
- Let's use the cheat sheet!

#### x86-64 Reference Sheet (GNU assembler format)

#### Instructions

#### Data movement

movq Src, Dest	Dest = Src
movsbq Src,Dest	Dest (quad) = Src (byte), sign-extend
movzbq Src,Dest	Dest (quad) = Src (byte), zero-extend

#### Conditional move

Equal / zero
Not equal / not zero
Negative
Nonnegative
Greater (signed $>$ )
Greater or equal (signed $\geq$ )
Less (signed $<$ )
Less or equal (signed $\leq$ )
Above (unsigned $>$ )
Above or equal (unsigned $\geq$ )
Below (unsigned $<$ )
Below or equal (unsigned $\leq$ )

#### Control transfer

cmpq Src2, Src1	Sets CCs Src1 Src2	
testq Src2, Src1	Sets CCs Src1 & Src2	
jmp label	jump	
je label	jump equal	
jne label	jump not equal	
js label	jump negative	
jns label	jump non-negative	
jg label	jump greater (signed $>$ )	
jge label	jump greater or equal (signed $\geq$ )	
jl label	jump less (signed <)	
jie label	jump less or equal (signed $\leq$ )	
ja label	jump above (unsigned $>$ )	
jb label	jump below (unsigned $<$ )	
pushq Src	%rsp = $%$ rsp 8, Mem[ $%$ rsp] = Src	
popq Dest	Dest = Mem[%rsp], %rsp = %rsp + 8	
call label push address of next instruction		
ret %rip = Mem[%rsp], %rsp = %rsp		

#### Arithmetic operations

leaq Src, Dest	Dest = address of Src
incq Dest	Dest = Dest + 1
decq Dest	Dest = Dest - 1
addq Src, Dest	Dest = Dest + Src
subq Src, Dest	Dest = Dest - Src
imulq Src, Dest	Dest = Dest * Src
xorq Src, Dest	$Dest = Dest ^ Src$
orq Src, Dest	Dest = Dest   Src
andq Src, Dest	Dest = Dest & Src
negq Dest	Dest = - Dest
notq Dest	$Dest = \sim Dest$
salq k, Dest	$Dest = Dest \ll k$
sarq k, Dest	$Dest = Dest \gg k$ (arithmetic)
shrq k, Dest	$Dest = Dest \gg k$ (logical)

#### Addressing modes

• Immediate Sval Val val: constant integer value movq \$7, %rax

Normal

(R) Mem[Reg[R]] R: register R specifies memory address movq (%rcx), %rax

Displacement

D(R) Mem[Reg[R]+D] R: register specifies start of memory region D: constant displacement D specifies offset movq 8(%rdi), %rdx

Indexed

label

D(Rb,Ri,S) Mem[Reg[Rb]+S\*Reg[Ri]+D] D: constant displacement 1, 2, or 4 bytes Rb: base register: any of 8 integer registers Ri: index register: any, except %esp S: scale: 1, 2, 4, or 8 movq 0x100(%rcx,%rax,4), %rdx

#### Instruction suffixes

- b byte
- word (2 bytes) W
- 1 long (4 bytes)
- quad (8 bytes) q

#### Condition codes

CF Carry Flag  $\mathbf{ZF}$ Zero Flag SF Sign Flag OF Overflow Flag

#### Integer registers

%rax Return value %rbx Callee saved %rcx 4th argument %rdx 3rd argument %rsi 2nd argument %rdi 1st argument %rbp Callee saved Stack pointer %rsp %r8 5th argument %r9 6th argument Scratch register %r10 %r11 Scratch register %r12 Callee saved %r13 Callee saved %r14 Callee saved %r15 Callee saved

### Control transfer

cmpq Src2, Src1	Sets CCs Src1 Src2
testq Src2, Src1	Sets CCs Src1 & Src2
jmp label	jump
je label	jump equal
jne label	jump not equal
js label	jump negative
jns label	jump non-negative
jg label	jump greater (signed $>$ )
jge label	jump greater or equal (signed $\geq)$

Src1 is %r10, Src2 is %r9
Set CCs based on Src1 <op>Src2, where <op>:=>

### cmpl %r9, %r10 jg 8675309

- So we jump if: %r10 > %r9
- "If the value of %r10 is greater than the value in %r9, then jump to 8675309"

## **Reading Assembly: Jumps**

Instruction	Condition	Description
jmp	1	Unconditional Jump
je/jz	ZF	Equal/Zero
jne/jnz	~ZF	Not Equal/Not Zero
js	SF	Negative
jns	~SF	Non-negative
ja	~ (SF^OF) &~ZF	Greater (Signed)
jge	~ (SF^OF)	Greater or Equal (Signed)
jl	(SF^OF)	Less (Signed)
jle	(SF^OF)  ZF	Less or Equal (Signed)
ja	~CF & ~ZF	Above (unsigned)
jb	CF	Below (unsigned)

## **Reading Assembly: Jumps**

cmp \$0x15213, %r12
jge deadbeef

cmp	%rax, %rdi
jae	15213b

If %**r12 >= 0x15213**, then jump to **0xdeadbeef**.

If the *unsigned* value in **%rdi** is greater than or equal to the *unsigned value* in **%rax**, jump to **0x15213b**.

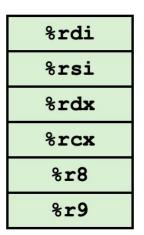
If %**r8** is not zero, jump to the address stored in %**rsi**.

## **Preview: Calling Conventions**

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

## **Calling Conventions: Passing Data**

How can we pass arguments to a procedure?



••• Arg n ••• Arg 8 Arg 7

First 6 arguments passed in *registers*.

Remaining arguments put at the end of the *caller's stack frame*.

## **Calling Conventions: Passing Data**

How can we access the return value?

%rax

Return value placed in %rax by convention.

## Bomblab

## **Bomblab: Premise**

- **Dr. Evil** has planted *binary bombs* on our shark machines!
- Your task: defuse your bomb by passing the correct strings on stdin.
- You get:
  - A C source file for the *main program*
  - An executable (no C source code for the phases!)
- Have to reverse engineer the bomb using only gdb and the assembly code!

## **Bomblab: Getting Started**

- Download your bomb from Autolab
- You must use the <u>Shark Machines</u> to extract (untar) and work on your Bomb.
- Run autolab setup
- 6 Progressively Harder Phases
  - Enter the correct string to move on to the next phase
  - Read the write up! It has an entire page dedicated to hints!

### Hints (Please read this!)

There are many ways of defusing your bomb. You can examine it in great detail without ever running the program, and figure out exactly what it does. This is a useful technique, but it not always easy to do. You can also run it under a debugger, watch what it does step by step, and use this information to defuse it. This is probably the fastest way of defusing it.

## **Bomblab: Detonating Your Bomb**

- Solving a phase automatically notifies Autolab and applies points to your score.
- If you let the bomb explode, Autolab will deduct 0.5 points each time.
  - Do <u>not:</u>
    - Use gdb to jump between phases
    - Solve the phases out of order
    - Tamper with the bomb
    - Otherwise the bomb will explode!

## **Bomblab: Defuse Kit**

## Defuse Kit: gdb

- **gdb** = GNU Debugger
- Fully-featured debugger:
  - For bomblab, lets you trace the execution of assembly
  - Useful for future labs, and well beyond 213.
  - Expand your debugging toolkit beyond printf!

## Defuse Kit: gdb

# Examining Program State print (p)

print \$rdi

Print contents of %rdi

(gdb) print /d 0x3b6d \$2 = 15213 Print with format

info

info registers

Print all register contents

### **x** (For eXamine)

- x /[num][size][format]
- **x** /**s** 0**x**... Examine contents of address as a string
- x /64bx 0x... View 64 bytes starting at the given address in Hex Format

## **GDB** Demo

If you want to follow along... (you'll also need this for the activity)

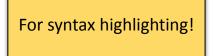
Download today's activity handout from the Schedule page.

```
$ wget http://www.cs.cmu.edu/~213/activities/s25-rec2.tar
$ tar xvpf s25-rec2.tar
$ cd s25-rec2
$ make
```

## **Defuse Kit: Getting the Assembly**

- Use **objdump** to get assembly code from your executable:
  - Then open and annotate in your favorite text editor!

```
objdump -d act1 > act1.asm
```



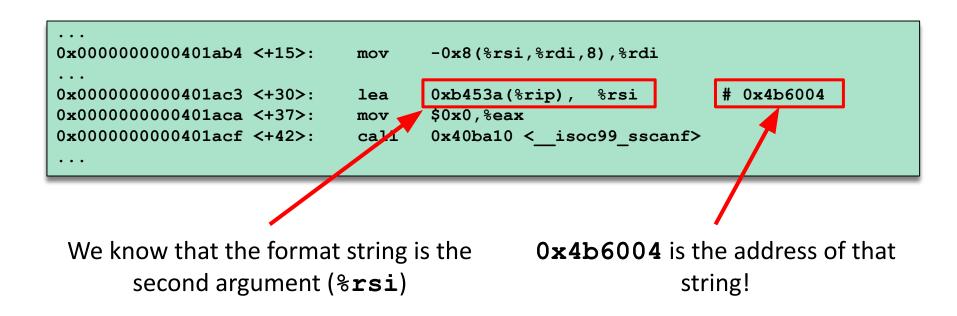
## Defuse Kit: Identifying inputs to main ()

- We see int main(int argc, char\*\* argv)
  - **main** is also a function we follow calling conventions
  - argc => %rdi, argv => %rsi
- Note that argv is a pointer type (array of arguments), meaning we must dereference to access the arguments!
  - Look out for addressing mode around %rsi

## **Defuse Kit: Figuring out Input Format**

Phases use **sscanf** to parse input strings:

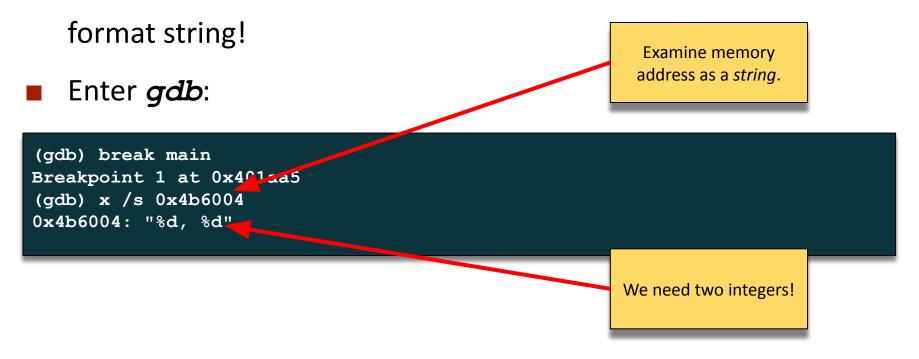
```
char *input_string = "123, 456";
int a, b;
sscanf(input_string, "%d, %d", &a, &b);
```



## **Defuse Kit: Figuring out Input Format**

•••		
0x000000000401ac3 <+30>:	lea	0xb453a(%rip), %rsi
0x0000000000401aca <+37>:	mov	\$0x0,%eax
0x0000000000401acf <+42>:	call	0x40ba10 < isoc99 sscanf>
		— –

If we can examine that memory address, we can recover the



## Warning: TUI Mode



### **TUI Mode**

- Is very cool (can view assembly alongside gdb prompt).
- But can unexpectedly *explode your bomb*.
- You will not get these points back.
- Can use vim/VSCode splitting instead.

## **GDB** Activity

## **GDB** Activity

- View the assembly and source code for act2
- Our objective is to match the source code to the assembly, identifying which sections correspond to each other!
- Get into groups of 3-4 and discuss together on how to interpret the assembly!
- If you understand the correlation fully along with the control flow in the assembly, feel free to try and solve the puzzle.