

Assembly and Bomb Lab

15-213: Introduction to Computer Systems

Recitation 4: Monday, Sept. 16, 2013

Marjorie Carlson

Section A

Agenda

- **Overview of Bomb Lab**
- Assembly Refresher
- Intro to GDB
- Unix Refresher
- Bomb Lab Demo



Bomb Lab

- Oh no! Dr. Evil has written an evil program that will “explode” the Shark machines!
- The program is in phases, each of which reads in input – something like a password – from standard input.
- If your input is correct, you go on to the next phase.
- If not, **the bomb explodes**. The program prints “BOOM!!!” and terminates, and you lose half a point. (Your score is updated **automatically** – you don’t have to upload anything to Autolab.)

Bomb Lab

- We give you:
 - Partial source code, in which Dr. Evil mocks you
 - The executable file itself
- You can't read the C source code. So how can you figure out what the program does?
- From the binary executable!

Agenda

- Overview of Bomb Lab
- **Assembly Refresher**
- Intro to GDB
- Unix Refresher
- Bomb Lab Demo



x86-64 Integer Registers

%rax	return	%eax
%rbx		%ebx
%rcx	arg 4	%ecx
%rdx	arg 3	%edx
%rsi	arg 2	%esi
%rdi	arg 1	%edi
%rsp		%esp
%rbp		%ebp

%r8	arg 5	%r8d
%r9	arg 6	%r9d
%r10		%r10d
%r11		%r11d
%r12		%r12d
%r13		%r13d
%r14		%r14d
%r15		%r15d

Assembly: Operands

Data type	Syntax	Examples	Notes
Immediate values (constant integers)	Start with \$	\$0x0 \$-15213	Don't forget 0x means hex!
Registers	Start with %	%esi %rax	Can represent a value or an address
Memory locations	Parentheses around a register, or addressing mode – D(Rb,Ri,S)	(%esi) 0x8(%rax) (%rax, %rsi, 4)	Parentheses dereference. If %esi stores an address, (%esi) is the value at that address.

Assembly: Some Common Operations

Instruction	Effect
<code>mov %rdi, %rax</code>	<code>rax = rdi</code>
<code>add %rdi, %rax</code>	<code>rax = rax + rdi</code>
<code>sub %rdi, %rax</code>	<code>rax = rax - rdi</code>
<code>lea (%rdi, %rsi, 2), %rax</code>	<code>rax = rdi + (2 * rsi)</code> (doesn't dereference)
<code>call foo</code>	Calls function "foo"
<code>push %eax</code>	Pushes <code>eax</code> onto the stack
<code>pop %eax</code>	Pops a value off the stack and into <code>eax</code>
<code>ret</code>	Returns to the return address (i.e., the next line in the calling function)
<code>nop</code>	Does nothing!
<i>You may see suffixes on the end: <code>b, w, l, q</code></i>	<i>Specify operand is 1, 2, 4, 8 bytes</i>

Assembly: Comparisons and Jumps

- Remember from class that Assembly uses comparisons and jumps (gotos) to execute various conditionals and loops.
- `cmp b, a` sets the same flags as computing $a - b$.
- `test b, a` sets the same flags as computing $a \& b$.
- These are usually followed by a conditional jump instruction that relies on the results.
- Watch out for operand order:

```
cmp1 %eax, %edx  
jg    401095
```



```
if %edx > %eax,  
    jump to 401095
```

Assembly: Comparisons and Jumps

Instruction	Effect	Instruction	Effect
jmp	Always jump	ja	Jump if above (unsigned >)
je/jz	Jump if =/0	jae	Jump if above or equal
jne/jnz	Jump if ≠/0	jb	Jump if below (unsigned <)
jg	Jump if >	jbe	Jump if below or equal
jge	Jump if >=	js	Jump if negative
j1	Jump if <	jns	Jump if nonnegative
jle	Jump if <=		

Assembly: Comparisons and Jumps

- `cmp $0x42, %edi`
`je 400d3b`
if `edi == 66`, jump to 400d3b
- `cmp %esi, %edx`
`jle 400e71`
if `edx <= esi`, jump to 400e71
- `test %rdi, %rdi`
`jne 400e87`
if `%rdi != 0`, jump to 400e87

Agenda

- Overview of Bomb Lab
- Assembly Refresher
- **Intro to GDB**
- Unix Refresher
- Bomb Lab Demo



Your Defusing Toolkit

- `objdump -t bomb` prints the symbol table
- `strings bomb` prints all printable strings
- `objdump -d bomb` prints the Assembly
- **`gdb bomb`** shows you the executable file in Assembly and lets you step through it line by line, peeking into the registers and stack as you go

All the GDB commands you need are in

<http://csapp.cs.cmu.edu/public/docs/gdbnotes-x86-64.pdf>

GDB: Stepping Through Code

■ **break** <location>

- sets a breakpoint. Location can be a function name or an address.
- Pro tip: you have to reset your break points when you restart GDB!

■ **run** / **run** <filename>

- runs the program up till the next breakpoint.
- Pro tip: instead of typing in your inputs each time, you can put them in a text file, one per line, and run that.

■ **disassemble** (or **disas** – but not **dis!!!**)

- shows you the current function, with an arrow to the **next** line.

■ **step** / **stepi** / **nexti**

- **step** executes one C statement – it doesn't work for us.
- **stepi** steps to the next line of Assembly.
- **nexti** does the same but doesn't stop in function calls.
- **stepi** <n> or **nexti** <n> steps through n lines.

GDB: Examining Data

■ **info registers**

- prints the (hex) contents of every register.

■ **print \$<register>**

- prints the contents of a register.
- Note the \$ – not a %.
- Use /x or /d, to specify hex or decimal: `print /d $rax.`

■ **x \$<register> / x 0x<address>**

- prints what the register points to (or what's at the given address).
- By default, prints one word (a “word” here is 4 bytes).
- However, in addition to specifying format (now including /s, string), you can specify how many objects of what size to print, in the format `x / [num][size][format]`, for example:
`x /4wd $rsp`

One Last Hint: `sscanf`

- The bomb frequently calls `sscanf` to read in formatted arguments.
- If you're not familiar with the formatting used by `printf`, now's the time!
- Example: `%s %x %s` represents an input of a string, hex number, and string.
- This could be handy in figuring out what kinds of arguments a phase is expecting.
- `man sscanf`!

Resources

- Assignment writeup
- GDB cheat sheet:
<http://csapp.cs.cmu.edu/public/docs/gdbnotes-x86-64.pdf>.
- CS:APP Chapter 3
- If you're stuck, check the course FAQ:
<http://www.cs.cmu.edu/~213/faq.html>
- If that doesn't help, email us: 15-213-staff@cs.cmu.edu
- Office hours: Sun-Thu, 5:30-8:30 pm, in Wean 5207
- Peer tutoring: Tue 8:30-11, Mudge Reading Room

- Note: if you Google Assembly instructions, make sure what you find is in **AT&T** syntax, not Intel. (The operands are reversed.)

Agenda

- Overview of Bomb Lab
- Assembly Refresher
- Intro to GDB
- **Unix Refresher**
- Bomb Lab Demo



Unix Refresher

- At the very least, you should be comfortable with:
 - `man` to read manual pages
 - `cd` to change directories
 - `ls` to list contents of the current directory
 - `ls -l` to list contents with extra info, including permission bits
 - `scp` to send files between your computer and the Shark machines
 - `ssh` to log into the Shark machines
 - `tar` to tar (`-cvf`) and untar (`-xvf`) things (`-z` for optional gzip)
 - `chmod` to change permission bits if necessary
 - flags (e.g. `-R` to apply a command recursively to a folder)
- Helpful hints: Tab autocompletes. An up arrow scrolls up through your last few commands.

Agenda

- Overview of Bomb Lab
- Assembly Refresher
- Intro to GDB
- Unix Refresher
- **Bomb Lab Demo**

