RUTGERS

THE STATE UNIVERSITY
OF NEW JERSEY

Recitation 7

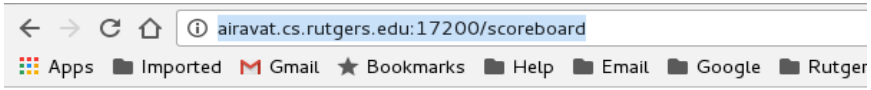
Jae Woo Joo

Programming Assignment 3

- Download bomb<N>.tar (N represents your ID)
  - http://ariavat.cs.rutgers.edu:17200
  - **Do not download more than 2 bombs!**
  - Download bomb in your iLab machine
- Untar your bomb
  - $ tar –xvf bomb<N>
  - Inside the bomb<N> directory it will have bomb, bomb.c, README
  - Only deal with the binary file **bomb**.
- Use GDB to solve
- Scores at http://airavat.cs.Rutgers.edu:17200/scoreboard
- Put your results in defuser.txt
- Submit your bomb along with the defuser.txt

Programming Assignment 3

- Scoreboard
  - You will lose 0.5 points for each explodes



Bomb Lab Scoreboard

This page contains the latest information that we have received from your bomb. If your solution is marked **invalid**, this means your bomb reported a solution that didn't actually defuse your bomb.

Last updated: Tue Mar 7 12:18:30 2017 (updated every 30 secs)

#	Bomb number	Submission date	Phases defused	Explosions	Score	Status
1	bomb3	Tue Feb 28 19:02	9	0	100	valid
2	bomb19	Sat Mar 4 18:50	9	0	100	valid
3	bomb15	Mon Mar 6 19:28	9	8	96	valid
4	bomb32	Sun Mar 5 06:25	8	8	86	invalid phase 9
5	bomb17	Mon Mar 6 23:07	6	0	60	invalid phase 7
6	bomb20	Tue Mar 7 11:27	7	1	75	invalid phase 8
7	bomb24	Sat Mar 4 20:56	5	0	45	invalid phase 6
8	bomb26	Sun Mar 5 09:23	5	1	45	invalid phase 6
9	bomb6	Thu Mar 2 19:21	5	33	29	invalid phase 6
10	bomb16	Mon Mar 6 20:39	4	4	33	invalid phase 5
11	bomb51	Mon Mar 6 23:51	3	0	25	invalid phase 4
12	bomb31	Sun Mar 5 14:17	3	1	25	invalid phase 4
13	bomb48	Tue Mar 7 00:06	3	1	25	invalid phase 4
14	bomb5	Tue Mar 7 08:00	2	14	8	invalid phase 3
15	bomb28	Sat Mar 4 21:10	1	5	3	invalid phase 2
16	bomb37	Sun Mar 5 14:43	0	1	0	invalid phase 1
17	bomb47	Mon Mar 6 08:47	0	1	0	invalid phase 1
18	bomb41	Sun Mar 5 19:47	0	2	-1	invalid phase 1
19	bomb44	Sun Mar 5 22:57	0	2	-1	invalid phase 1
20	bomb18	Sat Mar 4 15:11	0	3	-1	invalid phase 1
21	bomb30	Mon Mar 6 16:50	0	10	-5	invalid phase 1
22	bomb34	Mon Mar 6 22:11	0	17	-8	invalid phase 1
23	bomb61	Tue Mar 7 11:27	0	10266140	-40	invalid phase 1

Summary [phase:cnt] [1:1] [2:1] [3:3] [4:1] [5:3] [6:1] [7:1] [8:1] [9:3] total defused = 2/23

How to Defuse the bomb

- Debugging the bomb file using GDB
  - $ gdb bomb
  - Set break points for every phase (e.g. (gdb) break phase_1)
  - Run the program ((gdb) run)

- Useful commands for bomb
  - Print bomb's symbol table ($ objdump –t bomb)
  - Disassemble the code ($ objdump –d bomb)
  - Display printable strings ($ strings –t x bomb)

- You can save the instructions into the file
  - $ objdump –d bomb > (file_name).txt

How to Defuse the bomb

```
8048b6f:    e8 c0 09 00 00          call   8049534 <read_line>
8048b74:    89 04 24                mov    %eax,(%esp)
8048b77:    e8 04 01 00 00          call   8048c80 <phase_1>
8048b7c:    e8 ad 0a 00 00          call   804962e <phase_defused>
8048b81:    c7 04 24 40 a4 04 08    movl   $0x804a440,(%esp)
8048b88:    e8 f3 fc ff ff          call   8048880 <puts@plt>
8048b8d:    e8 a2 09 00 00          call   8049534 <read_line>
8048b92:    89 04 24                mov    %eax,(%esp)
8048b95:    e8 2a 01 00 00          call   8048cc4 <phase_2>
8048b9a:    e8 8f 0a 00 00          call   804962e <phase_defused>
8048b9f:    c7 04 24 81 a3 04 08    movl   $0x804a381,(%esp)
8048ba6:    e8 d5 fc ff ff          call   8048880 <puts@plt>
8048bab:    e8 84 09 00 00          call   8049534 <read_line>
8048bb0:    89 04 24                mov    %eax,(%esp)
8048bb3:    e8 30 01 00 00          call   8048ce8 <phase_3>
8048bb8:    e8 71 0a 00 00          call   804962e <phase_defused>
8048bbd:    c7 04 24 9f a3 04 08    movl   $0x804a39f,(%esp)
8048bc4:    e8 b7 fc ff ff          call   8048880 <puts@plt>
8048bc9:    e8 66 09 00 00          call   8049534 <read_line>
8048bce:    89 04 24                mov    %eax,(%esp)
8048bd1:    e8 9c 01 00 00          call   8048d72 <phase_4>
8048bd6:    e8 53 0a 00 00          call   804962e <phase_defused>
8048bdb:    c7 04 24 6c a4 04 08    movl   $0x804a46c,(%esp)
8048be2:    e8 99 fc ff ff          call   8048880 <puts@plt>
8048be7:    e8 48 09 00 00          call   8049534 <read_line>
8048bec:    89 04 24                mov    %eax,(%esp)
8048bef:    e8 d6 01 00 00          call   8048dca <phase_5>
8048bf4:    e8 35 0a 00 00          call   804962e <phase_defused>
8048bf9:    c7 04 24 b0 a3 04 08    movl   $0x804a3b0,(%esp)
8048c00:    e8 7b fc ff ff          call   8048880 <puts@plt>
8048c05:    e8 2a 09 00 00          call   8049534 <read_line>
8048c0a:    89 04 24                mov    %eax,(%esp)
```

Useful GDB Commands

- $ gcc –m32 hello.c –g –o hello
- $ gdb hello
- (gdb) run
- (gdb) c - continue
- (gdb) layout asm – gui for assembly code
- (gdb) ni – next instruction
- (gdb) si – step into the function
- (gdb) disas – disassemble instructions
- (gdb) until *addr – jump to given addr
- (gdb) i r – information for all registers
- (gdb) x/s (x/d) addr – print value of address
- More info
  - https://www.csee.umbc.edu/~cpatel2/links/310/nasm/gdb_help.shtml

LEAL instruction

- leal: compute address using addressing mode but does not use memory access

- leal src, dest
  - Src: Address mode expression (using parenthesis)
  - Dest: The address specified by src

- Usage
  - Computing address without using memory reference
  - E.g) p = &x[i]
  - leal 7(%edx, %edx, 4), %eax
    => eax = 4*edx + edx + 7

Shift Operations

- Left Shift: x << y
  - Shift bit-vector **x** left **y** positions
    - Throw away extra bits on left
    - Fill with 0's on right
- Right Shift: x >> y
  - Shift bit-vector **x** right **y** positions
    - Throw away extra bits on right
  - Logical shift (Unsigned)
    - Fill with 0's on left
  - Arithmetic shift (Signed)
    - Replicate most significant bit on left
- Undefined Behavior
  - Shift amount < 0 or ≥ word size

Argument **x**	**01100010**
<< 3	**00010*000***
Log. **>> 2**	***00***011000
Arith. **>> 2**	***00***011000

Argument **x**	**10100010**
<< 3	**00010*000***
Log. **>> 2**	***00***101000
Arith. **>> 2**	***11***101000

jaewoo.joo@cs.rutgers.edu

Shift operators

- sarl src, dest
  - Arithmetic right shift (dest = dest >> src)
- sall src, dest
  - Arithmetic left shift (dest = dest << src)
- shrl src dest
  - Logical right shift (dest = dest >> src)
- shll src dest
  - Logical left shift (dest = dest << src)

These two operates the same results

Stack

- Stacks grow toward lower addresses
- %esp is dedicated stack pointer register pointing to top of stack
- Push, pop instructions use $esp
- Push
  - Decrease %esp by 4
  - Copy operand value to where %esp points
- Pop
  - Copy top of stack to operand of pop instruction
  - Add 4 to %esp

Stack Operations

- %esp is used to maintain a stack in memory
- %esp contains the address of top of stack

- Push / pop
  - pushl %eax
    - esp = esp -4
    - Memory[esp] = eax
  - popl %ebx
    - ebx = Memory[esp]
    - esp = esp +4

Stack Example

- On the black board

Q & A

- Any questions?