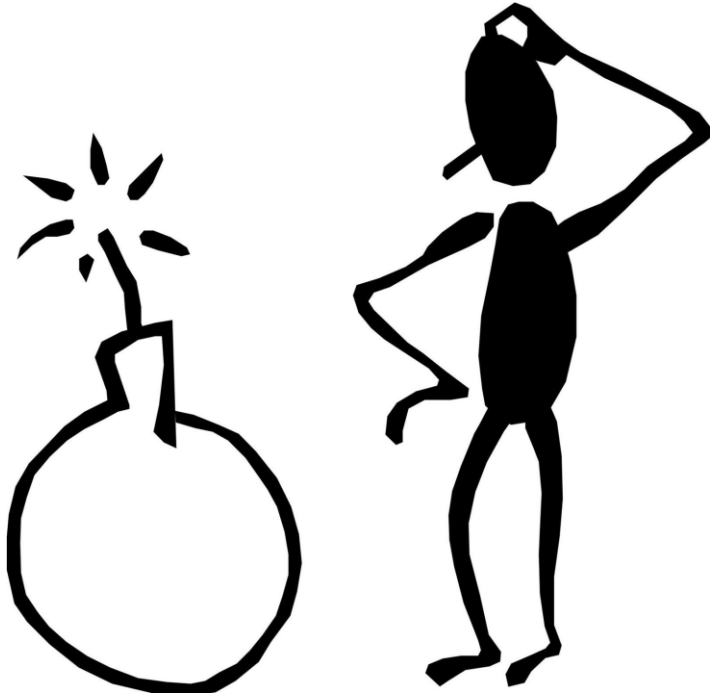
Bomb Lab

Instructors:

Sungyong Ahn

Agenda

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



What is Bomb Lab?

- An exercise in reading x86-64 assembly code.
- A chance to practice using GDB (a debugger).

- Why?
  - x86 assembly is low level machine code. Useful for understanding security exploits or tuning performance.
  - GDB can save you days of work in future labs and can be helpful long after you finish this class.

Downloading Your Bomb

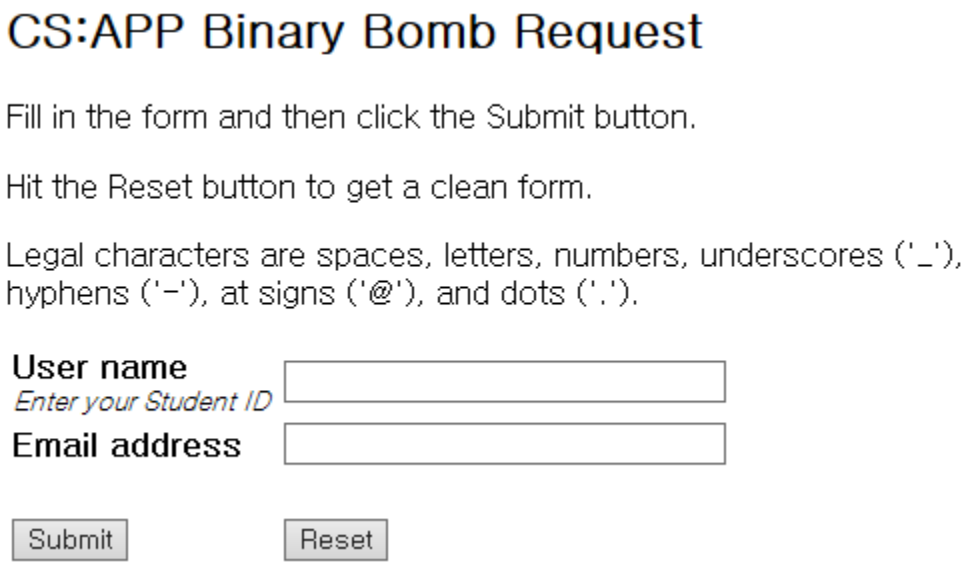
- All the details you'll need are in the write-up, which you most definitely have to read carefully before starting this lab anyway.
- ***Please Read The Writeup.***

Downloading Your Bomb

- Fine, here are some highlights of the write-up:
  - Bombs can only run on the `CSEDell` machines. They fail if you run them locally or on another server.
  - 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.

Get Your Bomb

- http://164.125.68.221:15217



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

Detonating Your Bomb

- Blowing up your bomb notifies server.
  - Dr. Evil takes **0.5** of your points each time.
  - It's very easy to prevent explosions using break points in GDB. More information on that soon.

```
jbiggs@makoshark ~/school/ta-15-213-f14/bomb170 $ ls
bomb  bomb.c  README
jbiggs@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.
jbiggs@makoshark ~/school/ta-15-213-f14/bomb170 $
```

Detonating Your Bomb

- Inputting the correct string moves you to the next phase.
- Don’t tamper with the bomb. Skipping or jumping between phases detonates the bomb.
- You have to solve the phases in order they are given. Finishing a phase also notifies server automatically.

Bomb Lab Scoreboard

- http://164.125.68.221:15217/scoreboard

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: Thu Oct 18 12:57:30 2018 (updated every 30 secs)

#	Bomb number	Submission date	Phases defused	Explosions	Score	Status
1	bomb2	Thu Oct 18 12:56	7	0	70	valid
2	bomb1	Thu Oct 18 12:57	3	6	27	valid

Summary [phase:cnt] [1:0] [2:0] [3:1] [4:0] [5:0] [6:0] [7:1] total defused = 1/2

Bomb Hints

- ***Dr. Evil*** may be evil, but he isn't cruel. You may assume that functions do what their name implies
  - i.e. phase_1() is most likely the first phase. printf() is just printf(). If there is an explode_bomb() function, it would probably help to set a breakpoint there!

- Use the man pages for library functions. Although you can examine the assembly for snprintf(), we assure you that it's easier to use the man pages ($ man snprintf) than to decipher assembly code for system calls.

A heavily redacted source file: bomb.c

```
/* Do all sorts of secret stuff that makes the bomb harder to defuse. */
   initialize_bomb();

   printf("Welcome to my fiendish little bomb. You have 6 phases with\n");
   printf("which to blow yourself up. Have a nice day!\n");

   /* Hmm...  Six phases must be more secure than one phase! */
   input = read_line();             /* Get input                   */
   phase_1(input);                  /* Run the phase               */
   phase_defused();                 /* Drat!  They figured it out!
                                     * Let me know how they did it. */
   printf("Phase 1 defused. How about the next one?\n");

   /* The second phase is harder.  No one will ever figure out
    * how to defuse this... */
   input = read_line();
   phase_2(input);
   phase_defused();
   printf("That's number 2.  Keep going!\n");
```

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	

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
`mov %rbx, %rdx`	rdx = rbx
`add (%rdx), %r8`	r8 += value at rdx
`mul $3, %r8`	r8 *= 3
`sub $1, %r8`	r8--
`lea (%rdx,%rbx,2), %rdx`	rdx = rdx + rbx*2

- *Doesn't dereference*

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
```

If _____, jump to addr 0xdeadbeef

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

If _____, jump to addr 0x15213b

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

If _____, jump to _____.

x64 Assembly: A Quick Drill

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

If `%r12 >= 0x15213`, 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`.

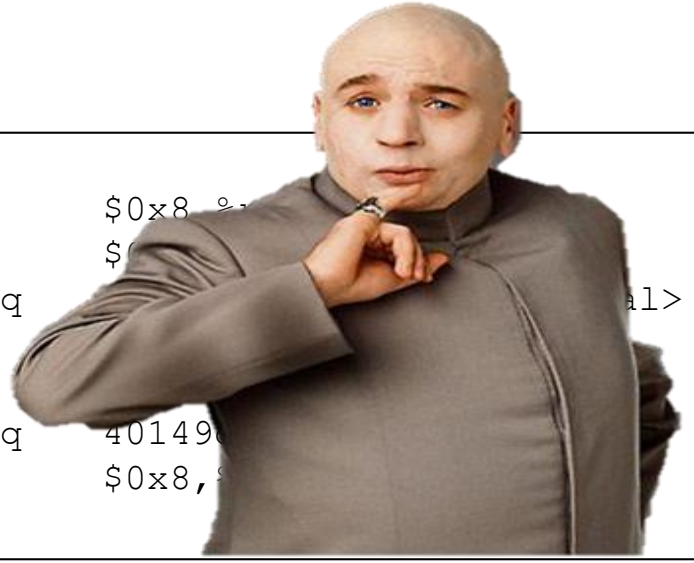
Diffusing Your Bomb

- `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 **G**NU **D**e**b**ugger
  - Examine while stepping through your program
    - registers
    - the stack
    - contents of program memory
    - instruction stream

Diffusing Your Bomb

objdump -d

```
0000000000400e8d <phase_1>:
  400e8d:  48 83 ec 08          sub    $0x8,%
  400e91:  be 30 24 40 00       mov    $
  400e96:  e8 03 05 00 00       callq                    al>
  400e9b:  85 c0                test
  400e9d:  74 05                je
  400e9f:  e8 f9 05 00 00       callq  40149
  400ea4:  48 83 c4 08          add    $0x8,
  400ea8:  c3                   retq
```



objdump -t

```
000000000040141d g   F .text  0000000000000002  initialize_bomb_solve
000000000040141f g   F .text  000000000000003d  blank_line
00000000004017d9 g   F .text  00000000000007fc  submitr
0000000000400f0d g   F .text  000000000000016d  phase_3
0000000000400e8d g   F .text  000000000000001c  phase_1
000000000040135b g   F .text  0000000000000025  invalid_phase
0000000000401ffd g   F .text  00000000000001d5  init_driver
0000000000000000     F *UND*  0000000000000000  alarm@@GLIBC_2.2.5
```

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

```
root@ubuntu:~/shared/bomb2# gdb bomb
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...done.
(gdb) break phase_1
Breakpoint 1 at 0x400f00
(gdb) run
Starting program: /media/sf_Shared/bomb2/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
qqq

Breakpoint 1, 0x0000000000400f00 in phase_1 ()
(gdb)
```

Using gdb

```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
=> 0x0000000000400f00 <+0>:     sub    $0x8,%rsp
   0x0000000000400f04 <+4>:     mov    $0x4024b0,%esi
   0x0000000000400f09 <+9>:     callq  0x401408 <strings_not_equal>
   0x0000000000400f0e <+14>:    test   %eax,%eax
   0x0000000000400f10 <+16>:    je     0x400f17 <phase_1+23>
   0x0000000000400f12 <+18>:    callq  0x40150a <explode_bomb>
   0x0000000000400f17 <+23>:    add    $0x8,%rsp
   0x0000000000400f1b <+27>:    retq
End of assembler dump.
(gdb) step
Single stepping until exit from function phase_1,
which has no line number information.

BOOM!!!
The bomb has blown up.
[Inferior 1 (process 1508) exited with code 010]
(gdb)
```

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 0xdeadbeef`

`sscanf`

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

Summary

- Download your own bomb and defuse it!
  - http://164.125.68.221:15217/
- 제출기한 : 11월 3일 11:59 PM
- 반드시 실습 서버에서 수행할 것!! (그 외의 환경에서는 실행 불가)
- 제출물
  - **BombID_학번.docx**: 각 phase에서 자신이 defusing code를 찾은 과정을 간략히 설명
    - MS워드 파일로 작성
    - 표지없이 간단히 첫 장 상단에 이름과 학번만 명시
    - 5장을 넘지 말 것, 초과시 보고서 점수 감점 사유.
    - PLMS로 제출

Hints!!

- Each bomb phase tests a different aspect of machine language programs:
  - Phase 1: string comparison
  - Phase 2: loops
  - Phase 3: conditionals/switches
  - Phase 4: recursive calls and the stack discipline
  - Phase 5: pointers
  - Phase 6: linked lists/pointers/structs

If you get stuck

- **Please read the writeup. *Please read the writeup*. *Please Read The Writeup.***
- View lecture notes for *Machine-Level Programming*
- If you have any questions, use the Q&A bulletin board on PLMS
- `man gdb, man sscanf, man objdump`