CENG 331 Rectitation 1
Defusing a Binary Bomb

Middle East Technical University
Department of Computer Engineering

October 23, 2019

Introduction

Bomblab Overview

Some Useful GDB Commands

Resources

Introduction

- Oh no!!! Dr. Evil placed an evil program in our INEK machines, we have to stop him!!! But how I wonder..?

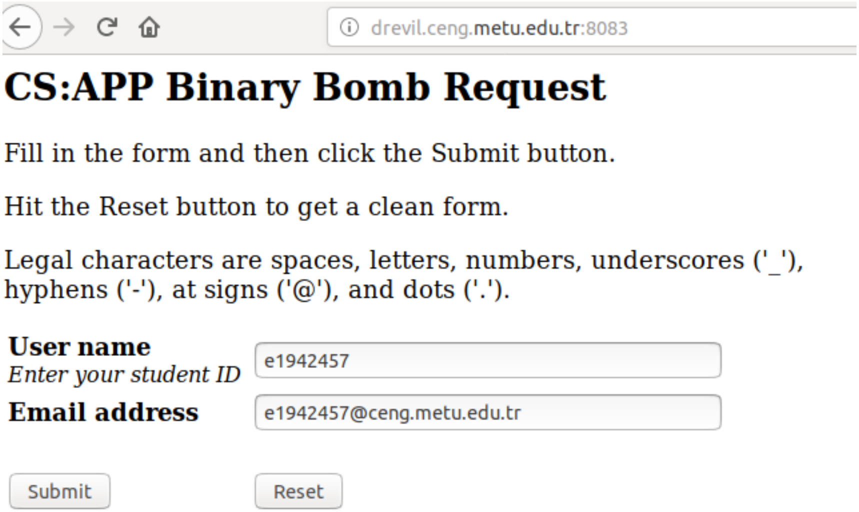
PURPOSE

- Make you familiar with assembly by hacking into some real programs.
- Understanding machine level operations.
- Improving debugging skills with the use of debuggers.

BOMBLAB

- You are handed a partial source code, in which Dr. Evil mocks you and your abilities as a hacker, and an executable file.
- You can't read the code but you need to figure out what it does. How!?
- From the binary executable itself!!!

Get Your Own Bomb



CS:APP Binary Bomb Request

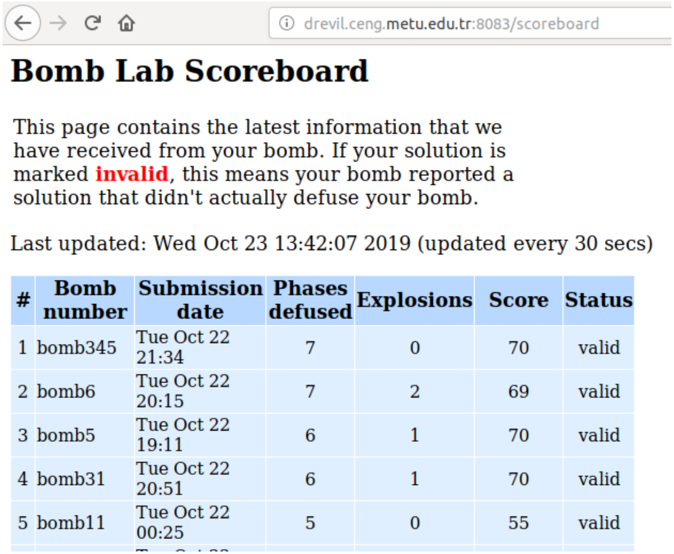
Fill in the form and then click the Submit button.

Hit the Reset button to get a clean form.

Legal characters are spaces, letters, numbers, underscores ('_'), hyphens ('-'), at signs ('@'), and dots ('.').

User name *Enter your student ID*	e1942457
Email address	e1942457@ceng.metu.edu.tr

SCOREBOARD



drevil.ceng.metu.edu.tr:8083/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: Wed Oct 23 13:42:07 2019 (updated every 30 secs)

#	Bomb number	Submission date	Phases defused	Explosions	Score	Status
1	bomb345	Tue Oct 22 21:34	7	0	70	valid
2	bomb6	Tue Oct 22 20:15	7	2	69	valid
3	bomb5	Tue Oct 22 19:11	6	1	70	valid
4	bomb31	Tue Oct 22 20:51	6	1	70	valid
5	bomb11	Tue Oct 22 00:25	5	0	55	valid

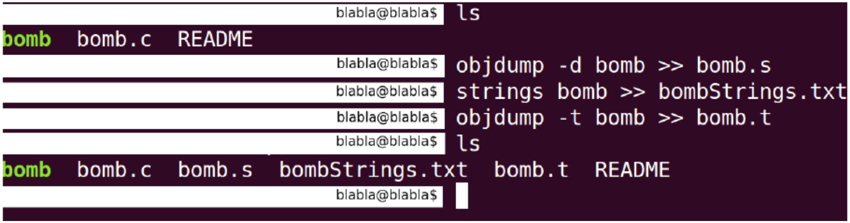
WHAT DOES IT DO

- Each students receives an executable with the name "bomb" and a C source code named "bomb.c".
- Executable expects strings from you. If you enter a wrong string it will explode, meaning that it will print a BOOM!!! message and send a signal to the server to decrement your points by 0.5.
- Every time you enter a correct string, you solve a phase and a new phase starts. After you solve 6 of them the bomb is defused.
- You can only work from inek machines. When you are outside of the department you can connect to the machines with SSH.

HOW TO START

- With objdump −d you can get the, relatively more readable, assembly code of the executable.
- objdump −d bomb >> bomb.s will write assembly code in bomb.s.
- You can search function names that are in bomb.c in the assembly file and trace what they do.
- With strings command you can print all strings in the executable into a file. Some of the strings may or may not lead you to some answers.

EXAMPLE



```
blabla@blabla$ ls
bomb   bomb.c   README
blabla@blabla$ objdump -d bomb >> bomb.s
blabla@blabla$ strings bomb >> bombStrings.txt
blabla@blabla$ objdump -t bomb >> bomb.t
blabla@blabla$ ls
bomb   bomb.c   bomb.s   bombStrings.txt   bomb.t   README
blabla@blabla$
```

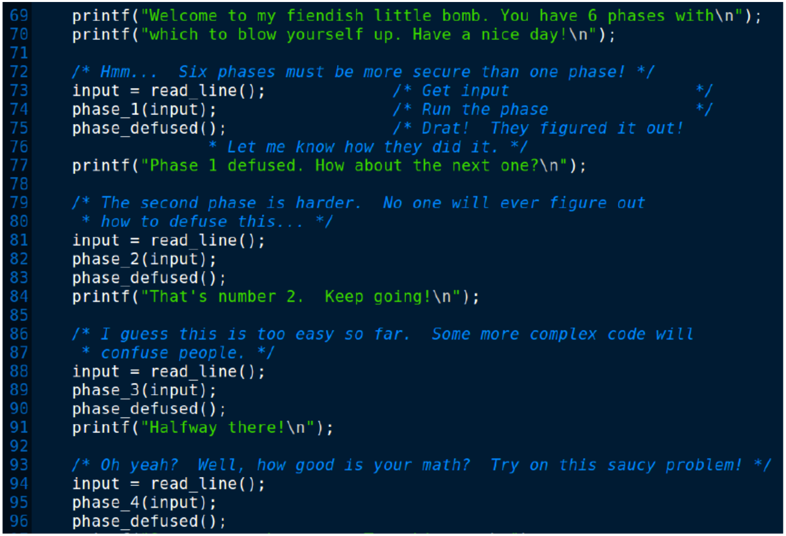
HINTS

- Make sure to run the bomb with GDB, this way you can put breakpoints to certain parts of the code to obstruct it from exploding.
- If you don't run the bomb in GDB you should enter all correct answers in one shot fashion to stop the bomb from exploding (Or you can really stop a bomb with Ctrl-C who knows?).
- You don't have to reenter all the strings you found to get to your current phase, you can feed them with "run solutions. txt" in GDB. "solutions.txt" is the file you saved your solutions up to that point.

GDB CRASH COURSE

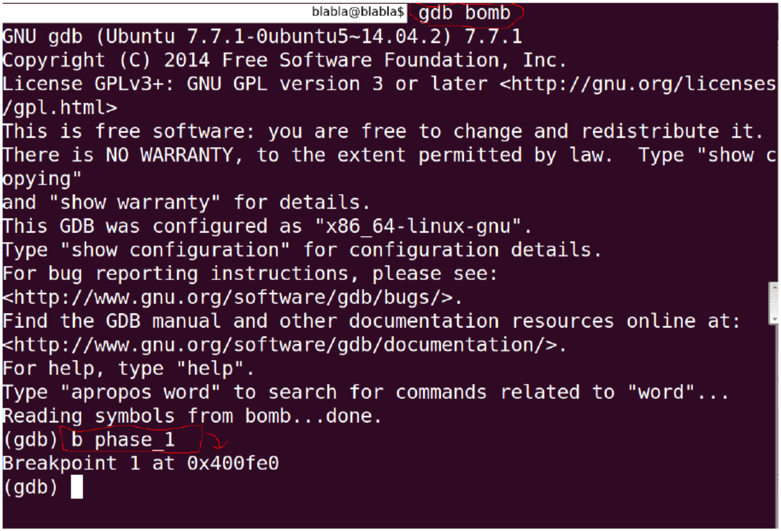
- GDB −> The GNU Project Debugger.
- Helps debugging the executable by running it line by line, putting breakpoints to instructions, examining memory content etc.
- Use it with an executable. Ex: ”gdb bomb”.

BOMB.C FILE



```
    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");

    /* I guess this is too easy so far.  Some more complex code will
     * confuse people. */
    input = read_line();
    phase_3(input);
    phase_defused();
    printf("Halfway there!\n");

    /* Oh yeah?  Well, how good is your math?  Try on this saucy problem! */
    input = read_line();
    phase_4(input);
    phase_defused();
```

EXAMPLE

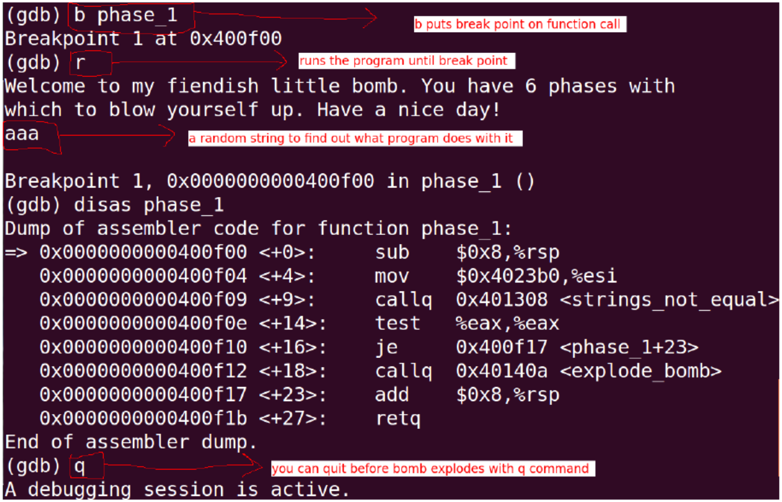


```
blabla@blabla$ gdb bomb
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses
/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show c
opying"
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) b phase_1
Breakpoint 1 at 0x400fe0
(gdb)
```

GDB Crash Course Cont'd

- b puts a break point.
- r runs the program until it hits a break point.
- disas prints the assembly code of a specific phase. You can also use the output of objdump to see the assembly code.

EXAMPLE



```
(gdb) b phase_1
Breakpoint 1 at 0x400f00
(gdb) r
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
aaa

Breakpoint 1, 0x0000000000400f00 in phase_1 ()
(gdb) disas phase_1
Dump of assembler code for function phase_1:
=> 0x0000000000400f00 <+0>:     sub    $0x8,%rsp
   0x0000000000400f04 <+4>:     mov    $0x4023b0,%esi
   0x0000000000400f09 <+9>:     callq  0x401308 <strings_not_equal>
   0x0000000000400f0e <+14>:    test   %eax,%eax
   0x0000000000400f10 <+16>:    je     0x400f17 <phase_1+23>
   0x0000000000400f12 <+18>:    callq  0x40140a <explode_bomb>
   0x0000000000400f17 <+23>:    add    $0x8,%rsp
   0x0000000000400f1b <+27>:    retq
End of assembler dump.
(gdb) q
A debugging session is active.
```

$\xrightarrow{}$ b puts break point on function call

$\xrightarrow{}$ runs the program until break point

$\xrightarrow{}$ a random string to find out what program does with it

$\xrightarrow{}$ you can quit before bomb explodes with q command

One Last Hint

- The bomb frequently calls sscanf to itemize your input strings.
- Ex: *"%s %x %s"* represents an input of a string, a hex number and a string.
- You can use this knowledge to figure out what kinds of arguments a phase is expecting.
- man sscanf !.

Resources

- A cheat sheet about GDB: http://csapp.cs.cmu.edu/3e/docs/gdbnotes-x86-64.pdf
- Chapter 3 from your book.
- Homework text. Read it carefully you can find many details there.
- You can use the discussion forum in Odtuclass for discussions.
- You can visit TA Erbil Yakışkan at A410 or send en email to: erbil@ceng.metu.edu.tr.