Defusing a Binary Bomb

System Program

Homework 4

2020.05.28

Introduction

Reverse Engineering

- Expect original source codes from binary files
  - Analyzing assembly codes
- Why do we need to reverse engineering?
  - Add new functionalities into non-open source program
    - Cracking commercial programs
    - Understanding unknown system logics
  - Security perspective
    - Analyzing malware
    - Mitigating cyber attacks

Abilities for Reverse Engineering

- You should know…
  - Basic architecture of Intel CPU
    - Registers, Little Endian, Instructions,…
  - Background about operating system
    - Virtual address, memory layout, file format,…
- Debugging skill
  - Debugging
    - Stop a program at any point and examine and change the values of variables line by line
  - Popular tool: `gdb`, `x96dbg`, `ollyDbg`,…

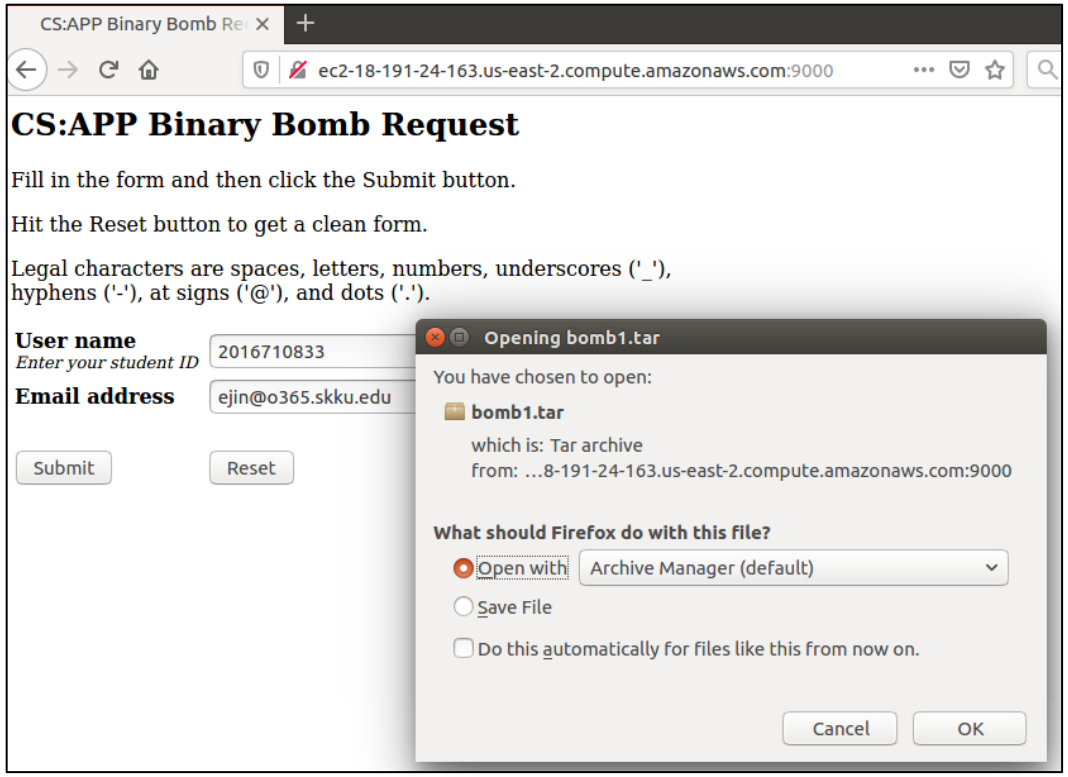
Homework 4:
Defusing a Binary Bomb

Homework #4

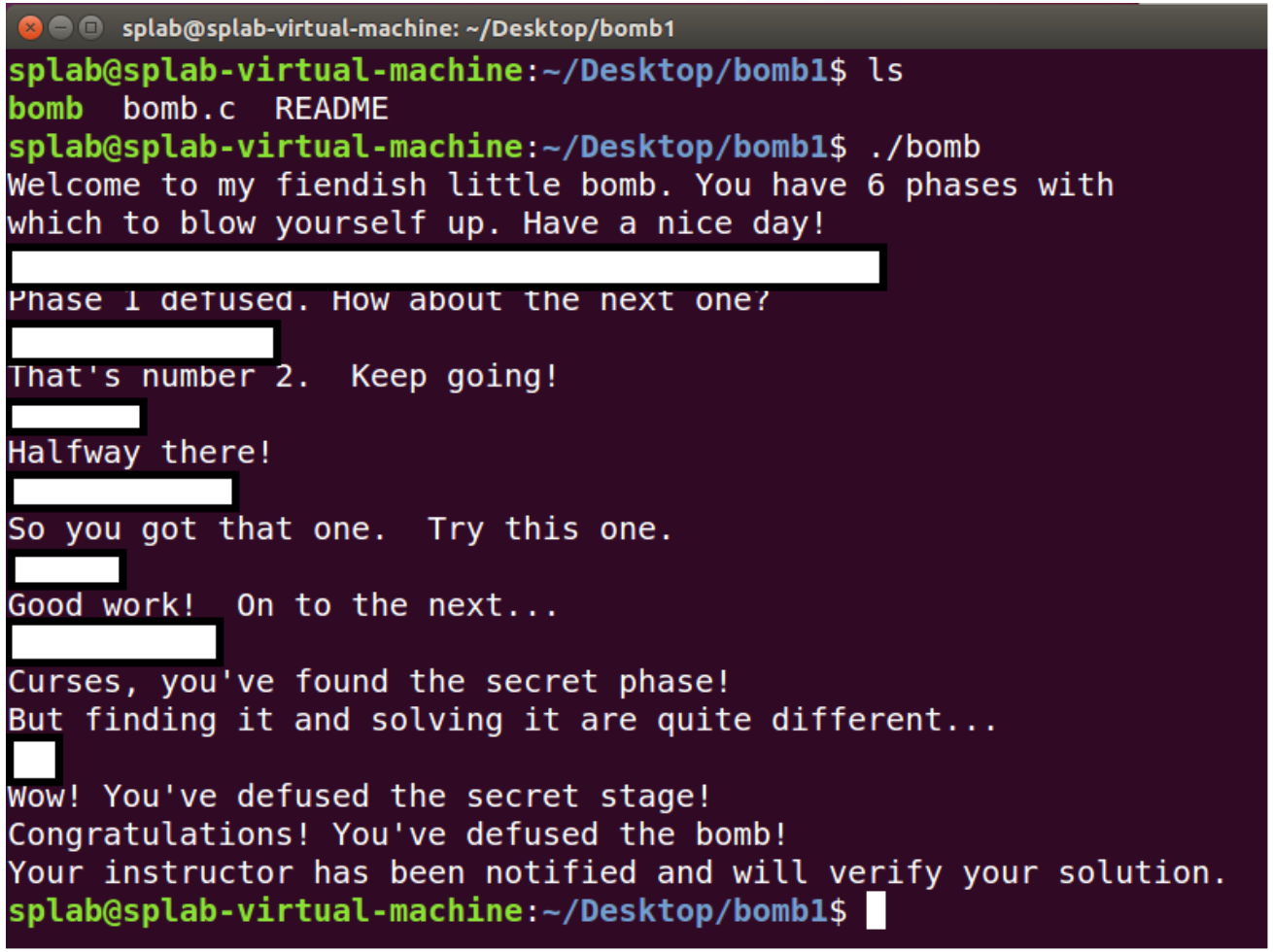
- Defusing a Binary Bomb
  - Get your bomb and defuse it
  - Type correct string, then the phase is defused and the bomb proceeds to the next phase
  - The bomb is defused when every phase has been defused
- Do your work on Linux
- Due date : 2020-06-05 (Friday) midnight

Get Your Bomb

- Binary bomb request server
  - http://ec2-18-191-24-163.us-east-2.compute.amazonaws.com:9000
- Enter your student ID and email address
  - Please enter valid student ID
- Then, you can get your BOMB!

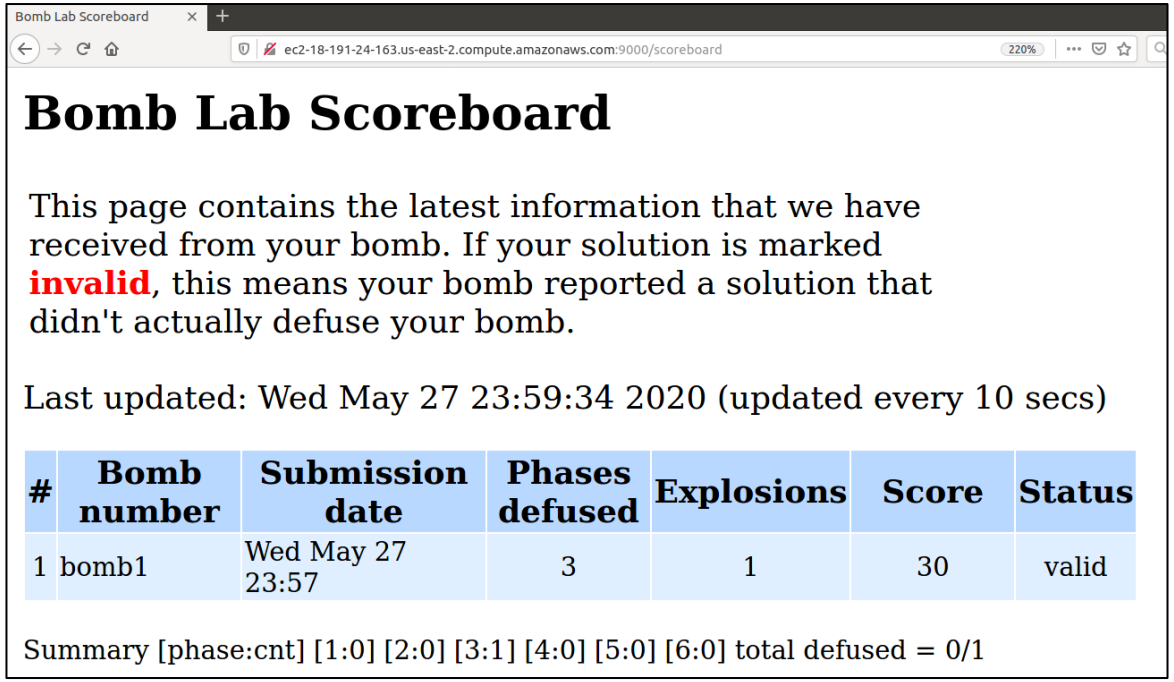


Defuse Your Bomb



Track Your Score

- Make sure your PC connect to the Internet
- Bomb lab scoreboard
  - http://ec2-18-191-24-163.us-east-2.compute.amazonaws.com:9000/scoreboard



Guides

bomb.c

- You can get hints about the bomb's logic
- Take inputs and determine which function it invokes
- Run the `phase_num(input)` function at each step

```
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();
printf("So you got that one.  Try this one.\n");

/* Round and 'round in memory we go, where we stop, the bomb blows! */
input = read_line();
phase_5(input);
phase_defused();
printf("Good work!  On to the next...\n");

/* This phase will never be used, since no one will get past the
 * earlier ones.  But just in case, make this one extra hard. */
input = read_line();
phase_6(input);
phase_defused();
```

GNU Project Debugger(gdb)

- Most popular debugger for Linux-based system
  - Command-line tool
- Important command
  - `r` : launch a program
  - `b [func. name]` : set a breakpoint at the beginning of a function
  - `ni` : step over
  - `si` : step into
  - `x/64dx [address]` : view memory value
  - `i r` : view register value
  - `disas [func. name]` : view assembly of a function