

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, o1lyDbg,...

Homework 4:

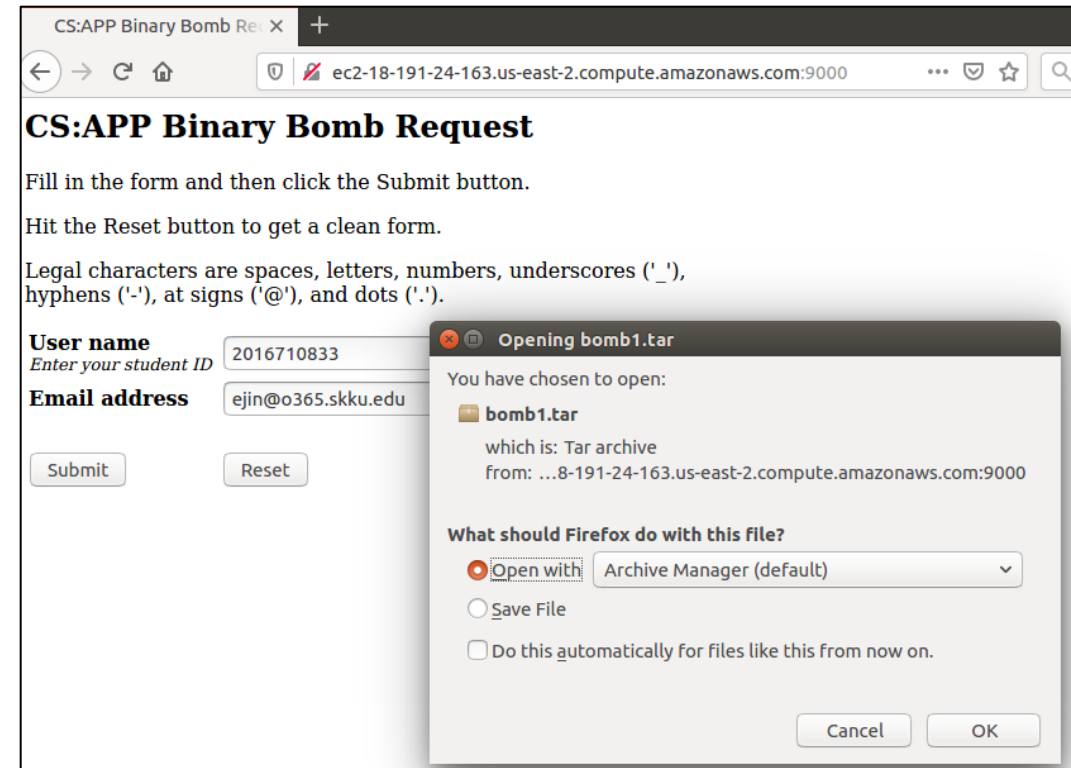
Defusing a Binary Bomb

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

```
splab@splab-virtual-machine: ~/Desktop/bomb1
splab@splab-virtual-machine:~/Desktop/bomb1$ ls
bomb  bomb.c  README
splab@splab-virtual-machine:~/Desktop/bomb1$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
_____
Phase 1 defused. How about the next one?
_____
That's number 2. Keep going!
_____
Halfway there!
_____
So you got that one. Try this one.
_____
Good work! On to the next...
_____
Curses, you've found the secret phase!
But finding it and solving it are quite different...
_____
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
splab@splab-virtual-machine:~/Desktop/bomb1$
```


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>

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 May 27 23:59:34 2020 (updated every 10 secs)

#	Bomb number	Submission date	Phases defused	Explosions	Score	Status
1	bomb1	Wed May 27 23:57	3	1	30	valid

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

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

```
splab@splab-virtual-machine: ~/Desktop/bomb1
splab@splab-virtual-machine:~/Desktop/bomb1$ clear
splab@splab-virtual-machine:~/Desktop/bomb1$ ./gdb
bash: ./gdb: No such file or directory
splab@splab-virtual-machine:~/Desktop/bomb1$ gdb bomb
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 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) disas main
Dump of assembler code for function main:
0x00000000400df6 <+0>:    push    %rbx
0x00000000400df7 <+1>:    cmp     $0x1,%edi
0x00000000400dfa <+4>:    jne     0x400e0c <main+22>
0x00000000400dfc <+6>:    mov     0x20398d(%rip),%rax      # 0x604790 <stdin@GLIBC_2.2.5>
0x00000000400e03 <+13>:   mov     %rax,0x2039a6(%rip)      # 0x6047b0 <infile>
0x00000000400e0a <+20>:   jmp     0x400e6f <main+121>
0x00000000400e0c <+22>:   mov     %rsi,%rbx
0x00000000400e0f <+25>:   cmp     $0x2,%edi
0x00000000400e12 <+28>:   jne     0x400e4e <main+88>
0x00000000400e14 <+30>:   mov     0x8(%rsi),%rdi
0x00000000400e18 <+34>:   mov     $0x402524,%esi
0x00000000400e1d <+39>:   callq   0x400c60 <fopen@plt>
0x00000000400e22 <+44>:   mov     %rax,0x203987(%rip)      # 0x6047b0 <infile>
0x00000000400e29 <+51>:   test    %rax,%rax
0x00000000400e2c <+54>:   jne     0x400e6f <main+121>
0x00000000400e2e <+56>:   mov     0x8(%rbx),%rcx
0x00000000400e32 <+60>:   mov     (%rbx),%rdx
0x00000000400e35 <+63>:   mov     $0x402526,%esi
0x00000000400e3a <+68>:   mov     $0x1,%edi
---Type <return> to continue, or q <return> to quit---
```