CS220 - Computer System II
Project 3

Due: 11/18/2017, 11:59pm

1 Introduction

In this project, you will diffuse a multi-phase 32 bit binary bomb. A binary bomb is a program that consists of a sequence of phases. Each phase expects a particular string as input which can either be provided through the terminal (stdin) or through a file. If you provide the correct string, the phase is diffused, and you move on to the next phase. Otherwise, the bomb explodes by printing BOOM and terminates. The bomb is diffused when all phases have been diffused. You are allowed to work in groups of no more than 2 people. *You are required to email your group membership to the TAs no later than 11/07/2017.*

2 Instructions

- BEFORE YOU BEGIN: After downloading the bomb from mycourses, transfer it to remote, and verify that the md5checksum `$ md5sum ./bomb` is: 575495e7424864973559ff900189159e.
- You can attempt the project alone or in groups of two students. There is no extra credit for attempting the project alone. If you do decide to work in pairs, TAs must be notified with your group membership before 7th Nov 2017, 11:59 pm. If the TAs dont receive your group information by then, you will undertake the project alone. Pick your group carefully. Each member of the group receives the same score.
- You can download the bomb from mycourses. The bomb comprises of 7 phases labeled phase0 through phase6. You are required to solve all phases in order to diffuse the bomb. Before running the bomb, make it executable by running `$ chmod +x bomb`. *You are required to diffuse the bomb on remote.cs.binghamton.edu.*
- Source code for the main function is given below:

```
int main( int argc , char  argv []) {
  FILE  fp ; if(argc > 1) {
  fp = fopen(argv[1],  r  );
  if(fp == NULL) {
    goto usage ret;
  } else {
    fp = stdin;
  }
```

```
9
    initialize ();
11
    phase0 (fp);
13  phase1 (fp);
    phase2 (fp);
15  phase3 (fp);
    phase4 (fp);
17  phase5 (fp);
    phase6 (fp);
19
    return 0;
21
usage ret :
23  printf( Usage: %s OR%s <file>\n , argv[0], argv[0]);
    return 1;
25 }
```

Each phase either explodes or prints a statement acknowledging that the phase has been successfully diffused.

- Inputs can either be provided through stdin or can be typed into a text file (each line is the input for 1 phase) and the file can be provided as a command-line argument.
- You can score a total of 100 points. Phases 0 and 1 are worth 10 points each. Phases 2, 3, 4 and 5 are worth 15 points each. Phase 6 is worth 20 points.

3 Submission

Include inputs to each phase in a single text file (input for each phase in a seaparate line). This file should work as an input to the bomb. All students of the group must submit the text file on mycourses.

4 How to proceed

There are many ways of defusing your bomb. You can examine it in great detail without ever running the program, and figure out exactly what it does. This is a useful technique,

but it not always easy to do. You can also run it under a debugger, watch what it does step by step, and use this information to defuse it. This is probably the fastest way of defusing it. There are many tools which are designed to help you figure out both how programs work, and what is wrong when they do not work. Here is a list of some of the tools you may find useful in analyzing your bomb, and hints on how to use them.

1. `gdb` This is perhaps the single most powerful tool available to crack this project. Here are some tips for using gdb.

   - set a breakpoint at the beginning of every phase to perform a closer examination. Use `disas` to disassemble the current function (although objdump may be more useful).
   - Set break points just before and after a call instruction to a function to examine the arguments and return values. This can be very useful. Note that in 32 bit, arguments are sent on the stack. Return value is stored in `eax` register. Examine them!
   - Set a breakpoint in the function that explodes and examine how the control reached there. Next, ask yourself, "how can I change the input so that the control does not get to this point?"
   - To set disassembly on gdb to intel syntax, use `set disassembly-flavor intel`.
   - Use `layout asm` to switch to assembly layout, and combine it with `ni` and `si` to step through instructions. Use `b *address` to break at an address and examine the values of registers and memory.
   - For other documentation, type "help" at gdb or "man gdb" or "info gdb". There is a lot of help on gdb online. A gdb cheatsheet can be found here: `http://darkdust.net/files/GDB\%20Cheat\%20Sheet.pdf`

2. `objdump -t` This command will print out the bombs symbol table. The symbol table includes the names of all functions and global variables in the bomb, the names of all the functions the bomb calls, and their addresses. You may learn something by looking at the function names; they've been honestly named in ways that are useful clues about their function.

3. `objdump -d -M intel` Use this to disassemble all of the code in the bomb. You can also just look at individual functions. Reading the assembler code can tell you how the bomb works. Although `objdump -d` gives you a lot of information, it does not

tell you the whole story. You need a debugger to know the runtime values of registers and memory.

4. `strings` This utility will display the printable strings in your bomb. In general this challenge is not simply about making your input string match an exact string stored inside the bomb- the string may be parsed as numbers which have to satisfy certain properties, so the string literals may not be so useful.

5. This lab is a popular lab in asm programming. A step-by-step example of diffusing a similar bomb can be found here: `http://zpalexander.com/binary-bomb-lab-phase-1/`.