CS 213, Fall 2002
Bomb Lab
Assigned: October 11, Due: Friday, November 1, 11:59PM

1 Introduction

The nefarious *Dr. Evil* has planted a slew of "binary bombs" on our machines. A binary bomb is a program that consists of a sequence of phases. Each phase expects you to type a particular string on *stdin*. If you type the correct string, then the phase is *defused* and the bomb proceeds to the next phase. Otherwise, the bomb *explodes* by printing `"BOOM!!!"` and then terminating. The bomb is defused when every phase has been defused.

There are too many bombs for us to deal with, so we are giving each group a bomb to defuse. Your mission, which you have no choice but to accept, is to defuse your bomb before the due date. Good luck, and welcome to the bomb squad!

Step 1: Get Your Bomb

Each group of students will attempt to defuse their own personalized bomb. Each bomb is a Linux binary executable file that has been compiled from a C program. To obtain your group's bomb, one (and only one) of the group members should send mail to

```
ics2002@grayling.cs.northwestern.edu
```

with the following `Subject` line:

```
bomb request fall02 user1@host1 [user2@host2]
```

where `user1@host1` and `user2@host2` are the email addresses of your group members. (The second email address is optional; don't actually type the brackets.) The bomb daemon will mail the same bomb to each email address on the `Subject` line and assign it to those group members. For example, if your group consists of `foo@cs` and `bar@cs`, then your `Subject` line should be:

```
bomb request fall02 foo@cs bar@cs
```

The bomb daemon sends you your bomb as a uuencoded tar file in the body of the email message. Depending on your email client, you can extract your bomb from the email message in one of two ways:

- Case 1: Most email clients will show the bomb as an attachment called `bombBOMBID.tar`, where BOMBID is a positive integer. In this case, save the attachment (not the email message!) to a file called `bombBOMBID.tar`, copy the file to a Linux machine, and then type

  ```
  linux>  tar xvf bombBOMBID.tar
  ```

  Your bomb will be in the file called `./bombBOMBID/bomb`.

- Case 2: If your email reader does not show the bomb as an attachment, save the entire email message as a text file called `foo.txt`, copy the file to a Linux machine using `scp` and then type

  ```
  linux>  uudecode foo.txt; tar xvf bombBOMBID.tar
  ```

  As before, your bomb will be in `./bombBOMBID/bomb`.

  Warning: Unix uses the ASCII newline character (`0xa`) to terminate text lines, while Windows uses the ASCII carriage return character (`0xd`) for this purpose. If you save your email message from Windows and then don't use the proper tools (SCP) to copy the file to Unix, then the text lines might be improperly terminated with Windows end-of-line characters. In this case, UUDECODE will fail with the following complaint:

  ```
  unix>  uudecode foo.txt
  uudecode: foo.txt: No 'end' line
  ```

  To avoid this, be sure to save your file as a text file, and then copy it to Unix using SCP, which automatically converts from Windows to Unix end-of-line characters. Or you can use a text editor to explicitly replace each Windows carriage return character with a Unix newline character.

Step 2: Defuse Your Bomb

Once you have received your bomb from the bomb daemon, save it in a secure directory. Your job is to defuse the bomb.

You can use many tools to help you with this; please look at the **hints** section for some tips and ideas. The best way is to use your favorite debugger to step through the disassembled binary.

Each time your bomb explodes it notifies the staff, and you lose 1/4 point (up to a max of 10 points) in the final score for the lab. So there are consequences to exploding the bomb. You must be careful!

Each phase is worth 10 points, for a total of 60 points.

The phases get progressively harder to defuse, but the expertise you gain as you move from phase to phase should offset this difficulty. However, the last phase will challenge even the best students, so please don't wait until the last minute to start.

The bomb ignores blank input lines. If you run your bomb with a command line argument, for example,

```
linux>  ./bomb psol.txt
```

then it will read the input lines from `psol.txt` until it reaches EOF (end of file), and then switch over to `stdin`. In a moment of weakness, Dr. Evil added this feature so you don't have to keep retyping the solutions to phases you have already defused.

To avoid accidently detonating the bomb, you will need to learn how to single-step through the assembly code and how to set breakpoints. You will also need to learn how to inspect both the registers and the memory states. One of the nice side-effects of doing the lab is that you will get very good at using a debugger. This is a crucial skill that will pay big dividends the rest of your career.

Logistics

You should work in a group of 2 people.

Any clarifications and revisions to the assignment will be emailed to the entire class.

You should do the assignment on the class machines. In fact, there is a rumor that Dr. Evil really is evil, and the bomb will always blow up if run elsewhere. There are several other tamper-proofing devices built into the bomb as well, or so they say.

Hand-In

There is no explicit hand-in. The bomb will notify your instructor automatically after you have successfully defused it. You can keep track of how you (and the other groups) are doing by looking at

`http://grayling.cs.northwestern.edu/bombstats.html`

This web page is updated continuously to show the progress of each group.

Hints *(Please read this!)*

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.

We do make one request, *please do not use brute force!* You could write a program that will try every possible key to find the right one. But this is no good for several reasons:

- You lose 1/4 point (up to a max of 10 points) every time you guess incorrectly and the bomb explodes.
- Every time you guess wrong, a message is sent to the staff. You could very quickly saturate the network with these messages, and cause the system administrators to revoke your computer access.

- We haven't told you how long the strings are, nor have we told you what characters are in them. Even if you made the (wrong) assumptions that they all are less than 80 characters long and only contain letters, then you will have 26^{80} guesses for each phase. This will take a very long time to run, and you will not get the answer before the assignment is due.

There are many tools which are designed to help you figure out both how programs work, and what is wrong when they don't 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.

- `gdb`

  The GNU debugger, this is a command line debugger tool available on virtually every platform. You can trace through a program line by line, examine memory and registers, look at both the source code and assembly code (we are not giving you the source code for most of your bomb), set breakpoints, set memory watch points, and write scripts. Here are some tips for using `gdb`.

  - To keep the bomb from blowing up every time you type in a wrong input, you'll want to learn how to set breakpoints.
  - The CS:APP Student Site at `http://csapp.cs.cmu.edu/public/students.html` has a very handy single-page `gdb` summary.
  - For other documentation, type "`help`" at the `gdb` command prompt, or type "`man gdb`", or "`info gdb`" at a Unix prompt. Some people also like to run `gdb` under `gdb-mode` in `emacs`.

- `objdump -t`

  This will print out the bomb's 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!

- `objdump -d`

  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 doesn't tell you the whole story. Calls to system-level functions are displayed in a cryptic form. For example, a call to `sscanf` might appear as:

  ```
  8048c36:  e8 99 fc ff ff  call   80488d4 <_init+0x1a0>
  ```

  To determine that the call was to `sscanf`, you would need to disassemble within `gdb`.

- `strings`

  This utility will display the printable strings in your bomb.

- `strace`

  This utility will show each system call made by your bomb as it executes.

Looking for a particular tool? How about documentation? Don't forget, the commands `apropos` and `man` are your friends. In particular, `man ascii` might come in useful. Also, the web may also be a treasure trove of information. If you get stumped, feel free to ask your TA for help.