Assignment 2

"Bomb Lab"

- Guideline -

System Program 2018 -2

Prof. Honguk Woo

Instruction

The nefarious *Dr. Evil* has planted a slew of “binary bombs” on our class 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 student 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

You can **obtain your bomb** by pointing your Web browser at:

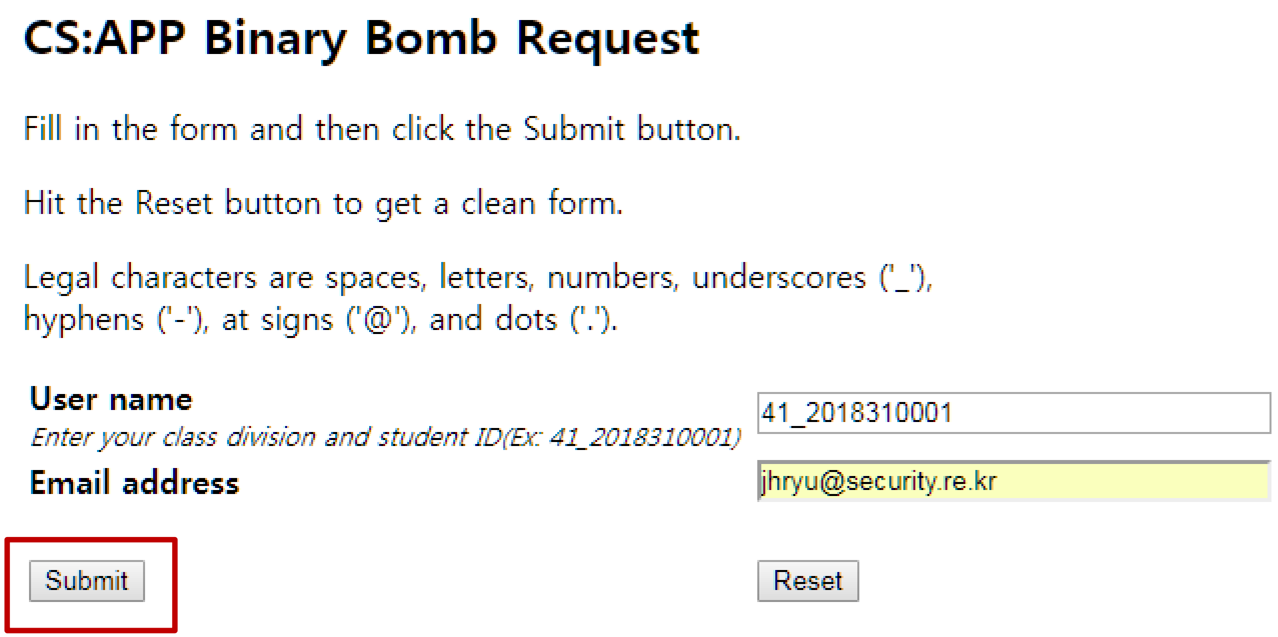
http://swin.skku.edu:10007/

This will display a binary bomb request form for you to fill in. Enter your user name and email address and **hit the Submit button**. The server will build your bomb and return it to your browser in a tar file called bombk.tar, where k is the unique number of your bomb.

Save the bombk.tar file to a (protected) directory in which you plan to do your work. Then give the command: tar -xvf bombk.tar. This will create a directory called ./bombk with the following files:

- **README** : Identifies the bomb and its owners.
- **bomb** : The executable binary bomb.
- **bomb.c** : Source file with the bomb's main routine and a friendly greeting from Dr. Evil.
- **writeup.{pdf,ps}** : The lab writeup.

Step 1 : Get Your 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 class division and student ID(Ex: 41_2018310001)*	41_2018310001
Email address	jhryu@security.re.kr

Step 2 : Defuse Your Bomb

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**
- **objdump –t**
- **objdump –d**
- **strings**

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.

The CS:APP web site

http://csapp.cs.cmu.edu/public/students.html

has a very handy single-page **gdb** summary that you can print out and use as a reference. Here are some other tips for using **gdb**.

Objdump

- **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**.

* https://linux.die.net/man/1/objdump

strings

For each file given, **strings** prints the printable character sequences that are at least 4 characters long (or the number given with the options below) and are followed by an unprintable character.

This utility will **display the printable strings** in your bomb.

* http://manpages.ubuntu.com/manpages/artful/man1/arm-linux-gnueabi-strings.1.html

How to Approach

< A list of useful functions in gdb>

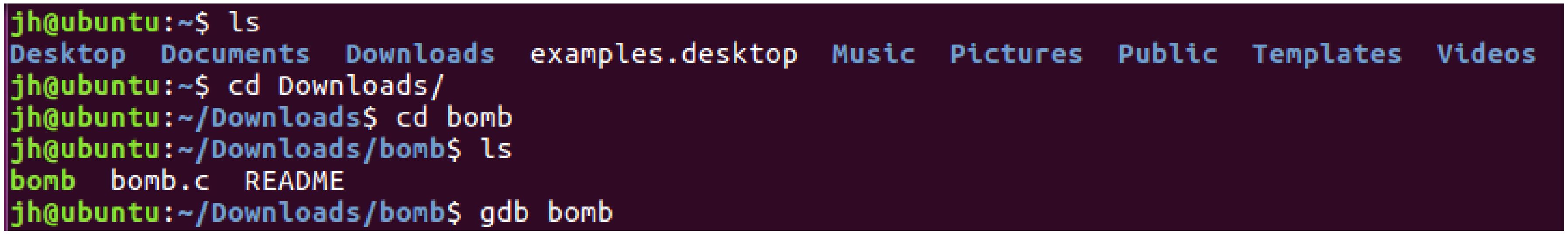
- **run** : run target program
- **break (function/address)** : set **breakpoint** on function or address
- **clear (function)** : clear **breakpoints** installed in the specified function
- **delete (breakpoint number)** : delete **breakpoint** of specified breakpoint number
- **Info**
    - **info reg** : output all register information
    - **info breakpoints** : output all breakpoints information
    - **info breakpoints (number)** : output specific breakpoint information

How to Approach

< A list of useful functions in gdb>

- **disas (function/address)** : disassemble on function or address
- **step / stepi/ nexti** : run until next command
- **cont** : run until next breakpoint
- **print (register)** : output information of register
- **x/~ (address)**
  - **x/x (address) :** output the value of specific address in hexadecimal
  - **x/s (address) :** output the value of specific address in string
  - **x/c (address)** : output the value of specific address in character

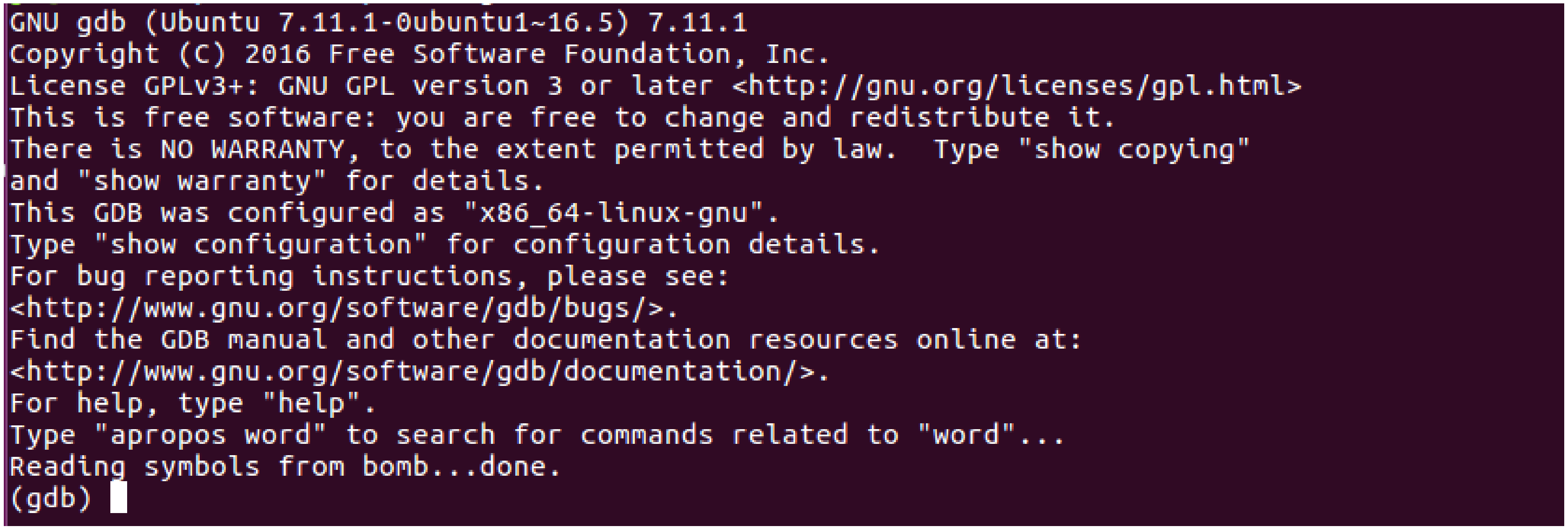
How to Approach



```
jh@ubuntu:~$ ls
Desktop  Documents  Downloads  examples.desktop  Music  Pictures  Public  Templates  Videos
jh@ubuntu:~$ cd Downloads/
jh@ubuntu:~/Downloads$ cd bomb
jh@ubuntu:~/Downloads/bomb$ ls
bomb  bomb.c  README
jh@ubuntu:~/Downloads/bomb$ gdb bomb
```

Frist, open the ‘**bomb**’ using **gdb**.

How to Approach



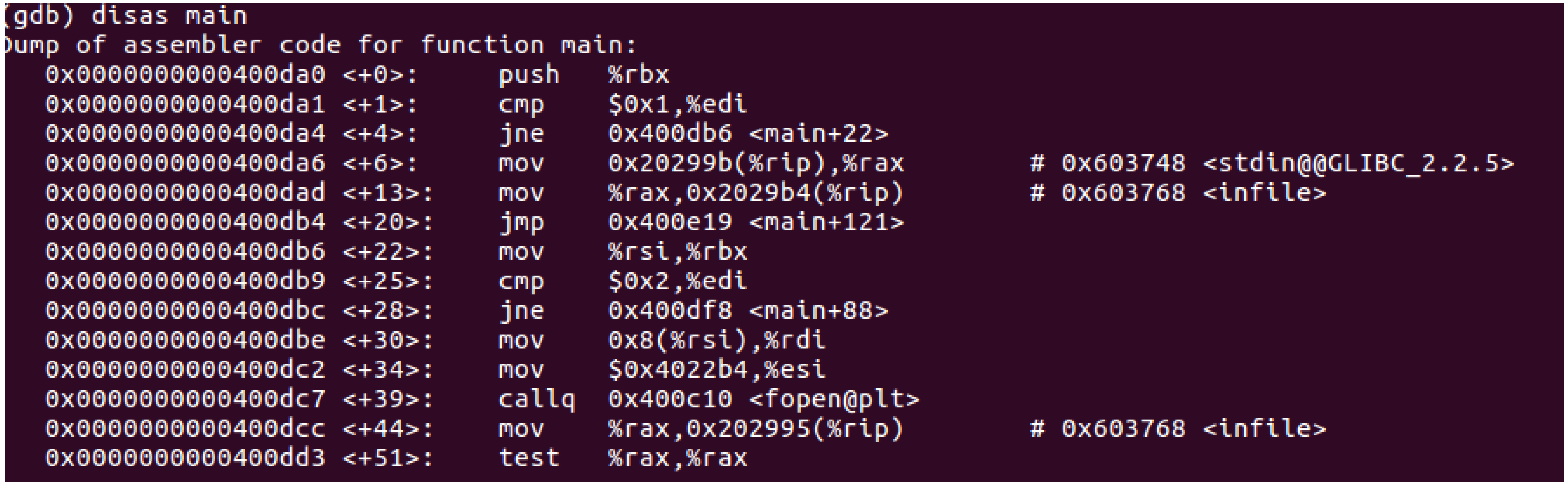
```
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "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) 
```

(gdb) input line will be appear.

Let's disassemble **main** here.

Type **'(gdb) disas main'**.

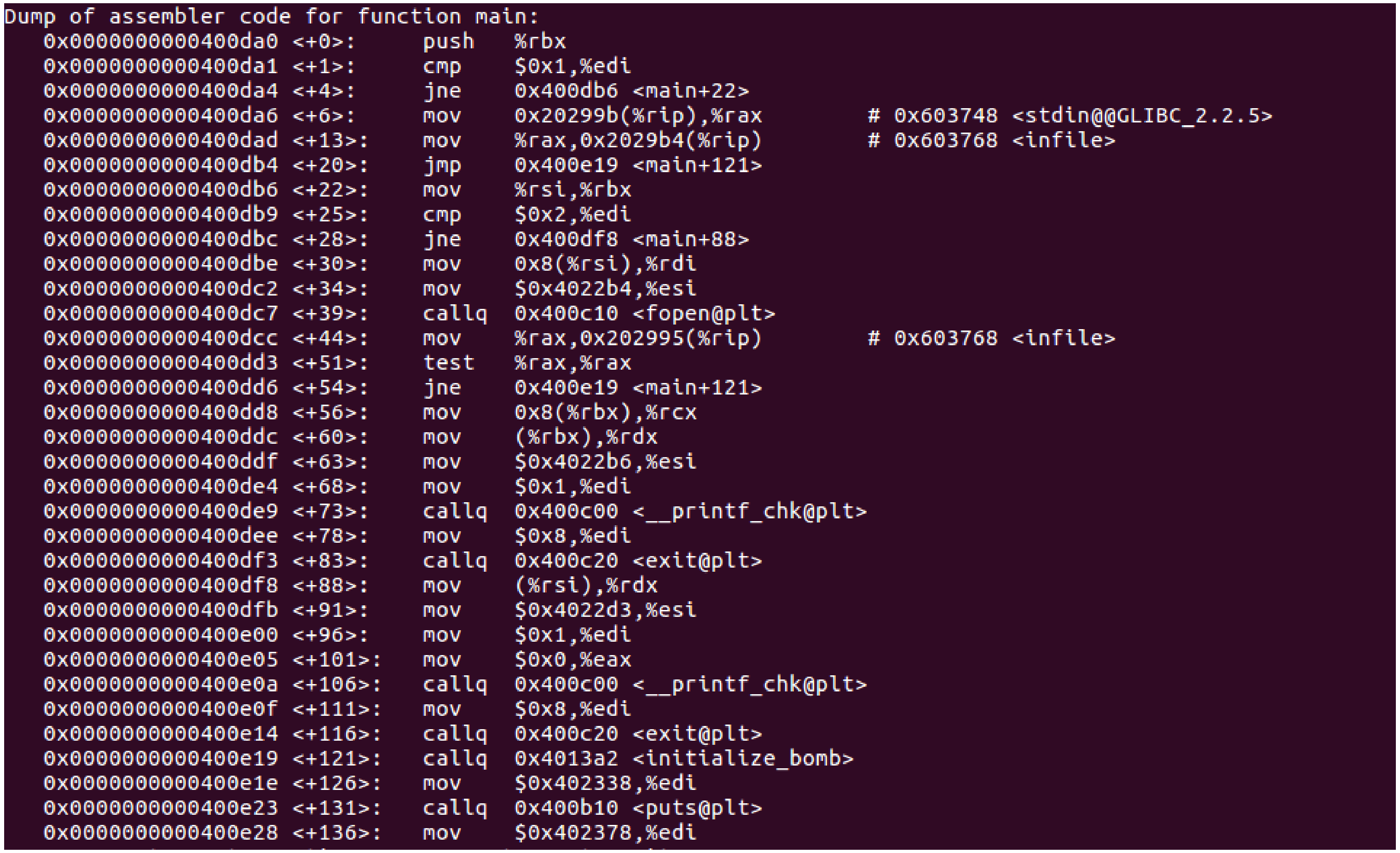
How to Approach



```
(gdb) disas main
Dump of assembler code for function main:
   0x0000000000400da0 <+0>:     push   %rbx
   0x0000000000400da1 <+1>:     cmp    $0x1,%edi
   0x0000000000400da4 <+4>:     jne    0x400db6 <main+22>
   0x0000000000400da6 <+6>:     mov    0x20299b(%rip),%rax        # 0x603748 <stdin@@GLIBC_2.2.5>
   0x0000000000400dad <+13>:    mov    %rax,0x2029b4(%rip)        # 0x603768 <infile>
   0x0000000000400db4 <+20>:    jmp    0x400e19 <main+121>
   0x0000000000400db6 <+22>:    mov    %rsi,%rbx
   0x0000000000400db9 <+25>:    cmp    $0x2,%edi
   0x0000000000400dbc <+28>:    jne    0x400df8 <main+88>
   0x0000000000400dbe <+30>:    mov    0x8(%rsi),%rdi
   0x0000000000400dc2 <+34>:    mov    $0x4022b4,%esi
   0x0000000000400dc7 <+39>:    callq  0x400c10 <fopen@plt>
   0x0000000000400dcc <+44>:    mov    %rax,0x202995(%rip)        # 0x603768 <infile>
   0x0000000000400dd3 <+51>:    test   %rax,%rax
```

As shown in the above, assembler code of ‘**main**’ will be appeared.

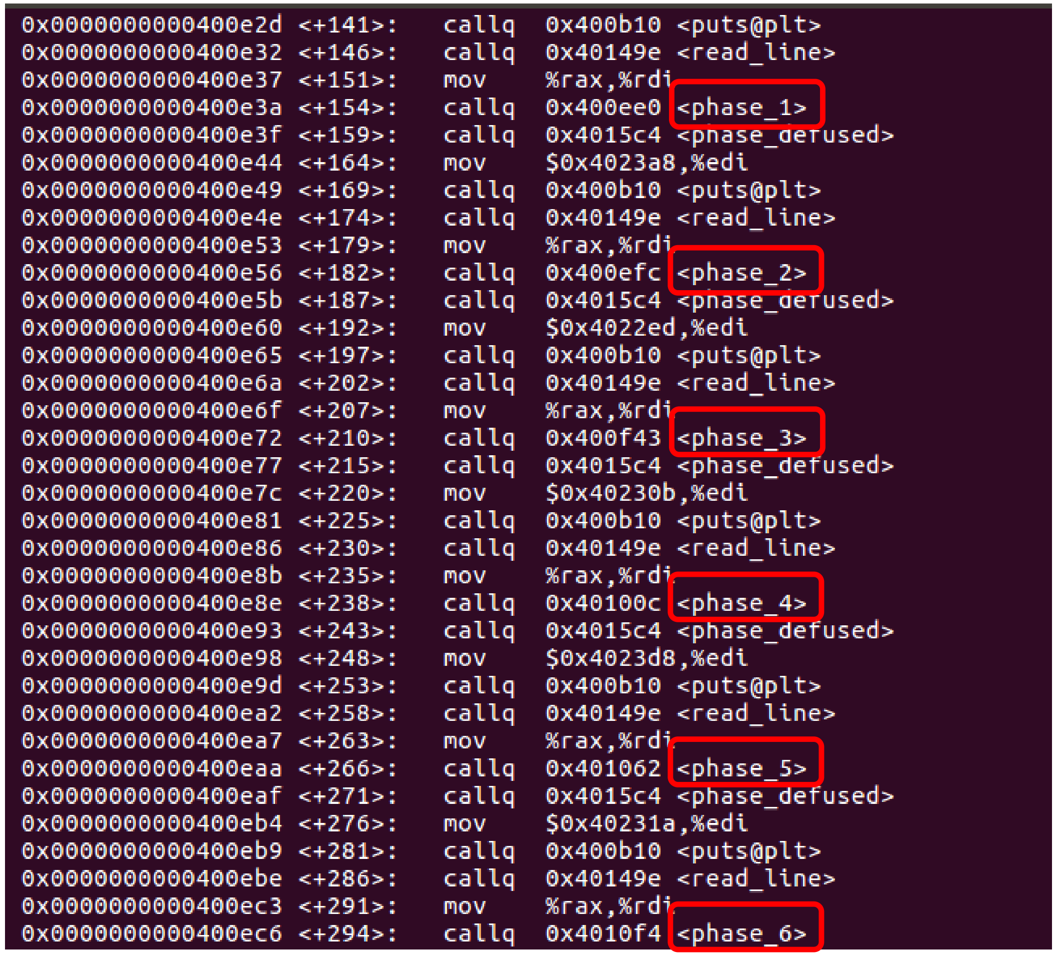
How to Approach



```
Dump of assembler code for function main:
   0x0000000000400da0 <+0>:     push   %rbx
   0x0000000000400da1 <+1>:     cmp    $0x1,%edi
   0x0000000000400da4 <+4>:     jne    0x400db6 <main+22>
   0x0000000000400da6 <+6>:     mov    0x20299b(%rip),%rax        # 0x603748 <stdin@@GLIBC_2.2.5>
   0x0000000000400dad <+13>:    mov    %rax,0x2029b4(%rip)        # 0x603768 <infile>
   0x0000000000400db4 <+20>:    jmp    0x400e19 <main+121>
   0x0000000000400db6 <+22>:    mov    %rsi,%rbx
   0x0000000000400db9 <+25>:    cmp    $0x2,%edi
   0x0000000000400dbc <+28>:    jne    0x400df8 <main+88>
   0x0000000000400dbe <+30>:    mov    0x8(%rsi),%rdi
   0x0000000000400dc2 <+34>:    mov    $0x4022b4,%esi
   0x0000000000400dc7 <+39>:    callq  0x400c10 <fopen@plt>
   0x0000000000400dcc <+44>:    mov    %rax,0x202995(%rip)        # 0x603768 <infile>
   0x0000000000400dd3 <+51>:    test   %rax,%rax
   0x0000000000400dd6 <+54>:    jne    0x400e19 <main+121>
   0x0000000000400dd8 <+56>:    mov    0x8(%rbx),%rcx
   0x0000000000400ddc <+60>:    mov    (%rbx),%rdx
   0x0000000000400ddf <+63>:    mov    $0x4022b6,%esi
   0x0000000000400de4 <+68>:    mov    $0x1,%edi
   0x0000000000400de9 <+73>:    callq  0x400c00 <__printf_chk@plt>
   0x0000000000400dee <+78>:    mov    $0x8,%edi
   0x0000000000400df3 <+83>:    callq  0x400c20 <exit@plt>
   0x0000000000400df8 <+88>:    mov    (%rsi),%rdx
   0x0000000000400dfb <+91>:    mov    $0x4022d3,%esi
   0x0000000000400e00 <+96>:    mov    $0x1,%edi
   0x0000000000400e05 <+101>:   mov    $0x0,%eax
   0x0000000000400e0a <+106>:   callq  0x400c00 <__printf_chk@plt>
   0x0000000000400e0f <+111>:   mov    $0x8,%edi
   0x0000000000400e14 <+116>:   callq  0x400c20 <exit@plt>
   0x0000000000400e19 <+121>:   callq  0x4013a2 <initialize_bomb>
   0x0000000000400e1e <+126>:   mov    $0x402338,%edi
   0x0000000000400e23 <+131>:   callq  0x400b10 <puts@plt>
   0x0000000000400e28 <+136>:   mov    $0x402378,%edi
```

As shown in the above, assembler codes of ‘**main**’ will be appeared.

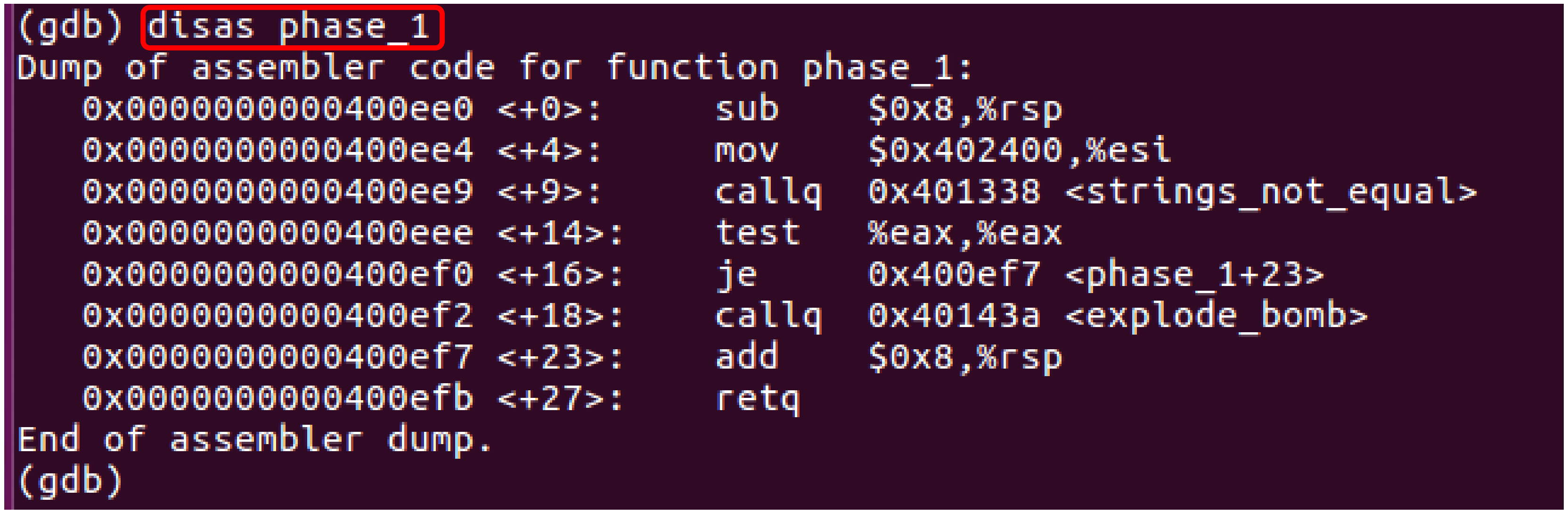
How to Approach



```
0x0000000000400e2d <+141>:   callq  0x400b10 <puts@plt>
0x0000000000400e32 <+146>:   callq  0x40149e <read_line>
0x0000000000400e37 <+151>:   mov    %rax,%rdi
0x0000000000400e3a <+154>:   callq  0x400ee0 <phase_1>
0x0000000000400e3f <+159>:   callq  0x4015c4 <phase_defused>
0x0000000000400e44 <+164>:   mov    $0x4023a8,%edi
0x0000000000400e49 <+169>:   callq  0x400b10 <puts@plt>
0x0000000000400e4e <+174>:   callq  0x40149e <read_line>
0x0000000000400e53 <+179>:   mov    %rax,%rdi
0x0000000000400e56 <+182>:   callq  0x400efc <phase_2>
0x0000000000400e5b <+187>:   callq  0x4015c4 <phase_defused>
0x0000000000400e60 <+192>:   mov    $0x4022ed,%edi
0x0000000000400e65 <+197>:   callq  0x400b10 <puts@plt>
0x0000000000400e6a <+202>:   callq  0x40149e <read_line>
0x0000000000400e6f <+207>:   mov    %rax,%rdi
0x0000000000400e72 <+210>:   callq  0x400f43 <phase_3>
0x0000000000400e77 <+215>:   callq  0x4015c4 <phase_defused>
0x0000000000400e7c <+220>:   mov    $0x40230b,%edi
0x0000000000400e81 <+225>:   callq  0x400b10 <puts@plt>
0x0000000000400e86 <+230>:   callq  0x40149e <read_line>
0x0000000000400e8b <+235>:   mov    %rax,%rdi
0x0000000000400e8e <+238>:   callq  0x40100c <phase_4>
0x0000000000400e93 <+243>:   callq  0x4015c4 <phase_defused>
0x0000000000400e98 <+248>:   mov    $0x4023d8,%edi
0x0000000000400e9d <+253>:   callq  0x400b10 <puts@plt>
0x0000000000400ea2 <+258>:   callq  0x40149e <read_line>
0x0000000000400ea7 <+263>:   mov    %rax,%rdi
0x0000000000400eaa <+266>:   callq  0x401062 <phase_5>
0x0000000000400eaf <+271>:   callq  0x4015c4 <phase_defused>
0x0000000000400eb4 <+276>:   mov    $0x40231a,%edi
0x0000000000400eb9 <+281>:   callq  0x400b10 <puts@plt>
0x0000000000400ebe <+286>:   callq  0x40149e <read_line>
0x0000000000400ec3 <+291>:   mov    %rax,%rdi
0x0000000000400ec6 <+294>:   callq  0x4010f4 <phase_6>
```

Looking closely at the code, you can find **<phase_1~6>.**

How to Approach



```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400ee0 <+0>:     sub    $0x8,%rsp
   0x0000000000400ee4 <+4>:     mov    $0x402400,%esi
   0x0000000000400ee9 <+9>:     callq  0x401338 <strings_not_equal>
   0x0000000000400eee <+14>:    test   %eax,%eax
   0x0000000000400ef0 <+16>:    je     0x400ef7 <phase_1+23>
   0x0000000000400ef2 <+18>:    callq  0x40143a <explode_bomb>
   0x0000000000400ef7 <+23>:    add    $0x8,%rsp
   0x0000000000400efb <+27>:    retq
End of assembler dump.
(gdb)
```

By using '(gdb) disas phase_1', you can see the assembler code for the <phase_1>.

Now is the time to show what you learned!!!

About Scores

Bomb Lab Scoreboard

- The bomb will notify your instructor automatically about your progress as you work on it. You can keep track of how you are doing by looking at the class scoreboard at:

http://swin.skku.edu:10007/scoreboard

You **lose ½ point** (up to a max of 20 points) every time you guess incorrectly and the bomb explodes

Every time you guess wrong, **a message is sent to the Bomb Lab server**. You could very quickly saturate the network with these messages, and cause the system administrators to revoke your computer access.

Q&A

<E-mail>

- 김종석 TA : ks77sj@gmail.com

- 유지현 TA : jhryu@security.re.kr

Please include “[SP]” in the title of your email.

e.g., “[SP] Assignment 2 질문입니다.”