CS 33 Fall 2023
Lab 2: Bomb Lab
Due: November 3rd, 11:59pm

Robert Chang (robertchang25@g.ucla.edu) is the lead person for this lab.

Introduction

A mysterious villain 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 all 6 phases have been defused.

There are too many bombs for us to deal with, so we are giving each student a bomb to defuse (don't question the morality of this decision). Your mission, if you choose to accept it (just kidding you have to accept it), is to defuse your bomb before the due date. Please take this spec and **read thoroughly** before starting. 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://cs33.seas.ucla.edu:15213/
```

Make sure you are connected to a UCLA network or using the Campus VPN service. The page will display a binary bomb request form for you to fill in. Enter your UID 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.

Upload bombk.tar file to a (protected) directory in which you plan to do your work. Please move your bomb to our servers using your favorite SCP file transfer GUI or from your terminal with

```
scp <yourlocalfile> <yourusername>@cs33.seas.ucla.edu:~/<yourdir>
```

Then use 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.

If for some reason you request multiple bombs, this is not a problem. Choose one bomb to work on and delete the rest. **We count explosions from any past bombs towards your final score.**

Step 2: Defuse Your Bomb

Your job for this lab is to defuse your bomb.

You must do the assignment on one of the class machines (Linux servers). In fact, there is a rumor that these bombs are particularly cruel, and the bomb will blow up if run elsewhere. There are several other tamper-proofing devices built into the bomb as well, or so we hear.

You can use many tools to help you defuse your bomb. 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 bomblab server, and you lose 1/2 point (up to a max of 20 points) in the final score for the lab. So there are consequences to exploding the bomb. You must be careful!

The first four phases are worth 10 points each. Phases 5 and 6 are a little more difficult, so they are worth 15 points each. So the maximum score you can get is 70 points.

Although phases get progressively harder to defuse, 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,

```
./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, it seems the perpetrator has added this feature so you don't have to keep retyping the solutions to phases you have already defused.

To avoid accidentally 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.

And as a final gift, you will be given 3 free explosions, no questions asked. Use them wisely :)

Logistics

This is an individual project. All handins are electronic and automatic. Clarifications and corrections will be announced on BruinLearn.

Handin

There is no explicit handin. 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://cs33.seas.ucla.edu:15213/scoreboard`

This web page is updated every 30 seconds to show the progress for each bomb.

Hints *(Please read this!)*

There are many ways of defusing your bomb. You can examine the assembly code 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.

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 1/2 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 bomblab server. 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 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 an extremely 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.
  - For online 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.

Another helpful aside, when going through the assembly code, you can safely step over any call to a function with a name prefixed with "__" (double underscore) or suffixed with "@plt"

Looking for a particular tool? How about documentation? Don't forget, the commands `apropos`, `man`, and `info` are your friends. In particular, `man ascii` might come in useful. `info gas` will give you more than you ever wanted to know about the GNU Assembler. If you get stumped, feel free to ask your TAs for help!

```
0000000000400f53 <phase_1>:
  400f53:       53                      push   %rbx
  400f54:       48 89 fb                mov    %rdi,%rbx
  400f57:       80 7f 01 00             cmpb   $0x0,0x1(%rdi)
  400f5b:       75 07                   jne    400f64 <phase_1+0x11>
  400f5d:       80 3b 6f                cmpb   $0x6f,(%rbx)
  400f60:       75 09                   jne    400f6b <phase_1+0x18>
  400f62:       5b                      pop    %rbx
  400f63:       c3                      retq
  400f64:       e8 31 07 00 00          callq  40169a <explode_bomb>
  400f69:       eb f2                   jmp    400f5d <phase_1+0xa>
  400f6b:       e8 2a 07 00 00          callq  40169a <explode_bomb>
  400f70:       eb f0                   jmp    400f62 <phase_1+0xf>
```

1) Push %rbx on to stack (-8 bits)

00 107f 08

%

2) Move %rdi into %rbx

phase 1 Answer: 'h'

1 | 2 = 3

1 2 4 8 5 6

hase_2 Disassembly

```
Dump of assembler code for function phase_2:
   0x0000000000400f72 <+0>:     push   %r12
   0x0000000000400f74 <+2>:     push   %rbp
   0x0000000000400f75 <+3>:     push   %rbx
   0x0000000000400f76 <+4>:     sub    $0x20,%rsp
   0x0000000000400f7a <+8>:     mov    %rsp,%rsi
   0x0000000000400f7d <+11>:    callq  0x40192d <read_six_numbers>
   0x0000000000400f82 <+16>:    mov    (%rsp),%eax
   0x0000000000400f85 <+19>:    or     0x4(%rsp),%eax
   0x0000000000400f89 <+23>:    je     0x400f95 <phase_2+35>
   0x0000000000400f8b <+25>:    mov    %rsp,%rbx
   0x0000000000400f8e <+28>:    lea    0x10(%rsp),%r12
   0x0000000000400f93 <+33>:    jmp    0x400fae <phase_2+60>
   0x0000000000400f95 <+35>:    callq  0x4016a8 <explode_bomb>
   0x0000000000400f9a <+40>:    jmp    0x400f8b <phase_2+25>
   0x0000000000400f9c <+42>:    mov    0x8(%rbp),%eax
   0x0000000000400f9f <+45>:    mov    %eax,0x20386b(%rip)        # 0x604810 <tr
ap>
   0x0000000000400fa5 <+51>:    add    $0x4,%rbx
   0x0000000000400fa9 <+55>:    cmp    %r12,%rbx
   0x0000000000400fac <+58>:    je     0x400fc2 <phase_2+80>
   0x0000000000400fae <+60>:    mov    %rbx,%rbp
   0x0000000000400fb1 <+63>:    mov    0x4(%rbx),%eax
```

pushing 3 variables in the stack

growing the stack by 4 elements

saves stack or return address

reads six #s

moves address of %rsp into %eax

CHECKPOINT 1: until *0x400f85

read_six_numbers disassembly

```
0x000000000040192d <+0>:     sub    $0x8,%rsp
0x0000000000401931 <+4>:     mov    %rsi,%rdx
0x0000000000401934 <+7>:     lea    0x4(%rsi),%rcx
0x0000000000401938 <+11>:    lea    0x14(%rsi),%rax
0x000000000040193c <+15>:    push   %rax
0x000000000040193d <+16>:    lea    0x10(%rsi),%rax
0x0000000000401941 <+20>:    push   %rax
0x0000000000401942 <+21>:    lea    0xc(%rsi),%r9
0x0000000000401946 <+25>:    lea    0x8(%rsi),%r8
0x000000000040194a <+29>:    mov    $0x402e4d,%esi
0x000000000040194f <+34>:    mov    $0x0,%eax
0x0000000000401954 <+39>:    callq  0x400ca0 <__isoc99_sscanf@plt>
0x0000000000401959 <+44>:    add    $0x10,%rsp
0x000000000040195d <+48>:    cmp    $0x5,%eax
0x0000000000401960 <+51>:    jle    0x401967 <read_six_numbers+58>
0x0000000000401962 <+53>:    add    $0x8,%rsp
0x0000000000401966 <+57>:    retq
0x0000000000401967 <+58>:    callq  0x4016a8 <explode_bomb>
```

* push %r12 pushes callee-saved temporary %r12 on to stack
* push %rbp pushes the frame pointer on to the stack.
* push %rbx pushes the callee-saved temporary %rbx on to stack.
* sub $0x20, %rsp subtracts 32 from %rsp and stores it into %rsp.
  Moves %rsp 4 memory slots down.
* mov %rsp, %rsi copies the value stored in %rsp into %rsi.

$\boxed{\%rsp} \longrightarrow \boxed{\%rsi}$

* call 0x401920 pushes return address address 0x400f7d on to stack and jumps to 0x401920

Call 0x401920 <read_six_numbers>

* sub $0x8, %rsp grows stack by 1 frame.
* mov %rsi, %rax moves contents of %rsi into %rax. %rax contains the address back to the return address
* lea 0x4(%rsi), %rcx copies the contents of %rsi + 4 into %rcx

```
  f f f
  0x400f7d
+       4
-----------
%rcx = 0x400f81
```

* lea 0x14(%rsi), %rax. Copies address of %rsi + 20 into %rax

```
  0x400f7d
+       14
-----------
%rax = 0x400f91
```

* push %rax pushes address value.
* lea 0x10(%rsi), %rax copies address of %rsi + 0x10 into %rax

(new) %rax = 0x400f8d

* push %rax pushes (new) %rax on to stack
* lea 0xc(%rsi), %rdi moves %rsi + 0xc onto %rdi register

```
%rdi = 0x400f7d
+           c
-----------
0x400f89
```

```
13
+12
--
25
```

* lea 0x8(%rsi), %r8

```
%r8 = 0x400f7d
+           8
-----------
0x400f85
```

* mov $0x402e8d, %esi

%esi
0x402e4d

* mov $0x0, %eax

%rax 0x400f8d
%eax 0

--- 0x4a4 --- sscanf@plt

* jmpq *0x20340a(%rip). Jumps to 0x6040aa

```
0x  400a0
    20340a
-----------
    6040aa
```

* push 0x13 pushes 0x13 onto stack
* jmpq 0x400b6b. Jump to 0x400b6b address.

} reads the inputted string

Call 0x401920 <read_six_numbers>

* add 0x10, %rsp shrinks stack by 2 frames.
* cmpq $0x5, %eax, %eax - 5 = -5
* If %eax is less than or equal to 5, bomb explodes.
  jumps to explode_bomb if address of return address is less than 0x401b98. Aim to have it higher than 0x401b98

Phase_2

* mov (%rsp), %eax copies address of memory address into %eax

%rax	0x400f7d	%eax

* %eax = %rsp+0x4 | %eax

```
= 0x400f81 | 0x400f7d
= 0100000000001111 0000001
| 0100000000001111 0111101
-------------------------
  0100000000001111 1111101
```

* mov %rsp, %rbx

%rbx 0x400f7d

* lea 0x10(%rsp), %r12

%r12 0x400f8d

Rules

- %r input string is in %rsi
- %r, esi is the last symbol in the string

Requirements

Last character must be greater than 5

Password Checks

- Must be a number
- Must contain 6 numbers
- Must have address greater than
- %eax must be greater than 5

Requirements

Last character must be greater than 5

* jmp 0x400fac. Jumps to 0x400fac

* mov %rbx, %rbp

%rbp	0x400f7d

```
  400f7d
+      4
--------
  400f81
```

* mov 0x4(%rbx), %eax

%rbx+4 ⟶ | %rbx 0x400f7d | %eax | 0x400f81 |

* add (%rbx), %eax

%eax = (%rbx) + %eax ⟶ | %rbx 0x400f7d | %eax | 001EFE |

* cmp %eax, 0x4(%rbx)

%eax	%0x4(%rbx)
001EFE	400F85

* je 0x400f9c . jump to address if %eax and %0x4(%rbx) are equal.

↳ If its not equal, the bomb explode)

* mov 0x4(%rsp), %eax

%rsp+4 ⟶ %eax

%rsp	400f7d	%eax	400F85

* mov %eax, 0x203466(%rip)

%eax ⟶ %rip + 0x 203466

400F85 0x400f9f + 0x203466 = 604040A

604040A ⟶ 400F85

* we are okay until 0x400f82 mov (%rsp), %eax

Answer: 1 2 3 5 8 13

* we are okay until 0x400f80

* 3

%r12-5

%rbx

```
Dump of assembler code for function phase_3:
   0x0000000000401016 <+0>:     push   %rbx
   0x0000000000401017 <+1>:     sub    $0x10,%rsp
   0x000000000040101b <+5>:     movl   $0x0,0xc(%rsp)
   0x0000000000401023 <+13>:    movl   $0x0,0x8(%rsp)
   0x000000000040102b <+21>:    lea    0x8(%rsp),%rcx
   0x0000000000401030 <+26>:    lea    0xc(%rsp),%rdx
   0x0000000000401035 <+31>:    mov    $0x402e59,%esi
   0x000000000040103a <+36>:    mov    $0x0,%eax
   0x000000000040103f <+41>:    callq  0x400ca0 <__isoc99_sscanf@plt>
CP1 0x0000000000401044 <+46>:    cmp    $0x1,%eax
   0x0000000000401047 <+49>:    jle    0x401053 <phase_3+61>
   0x0000000000401049 <+51>:    mov    0x8(%rsp),%eax
CP2 0x000000000040104d <+55>:    cmp    %eax,0xc(%rsp)
   0x0000000000401051 <+59>:    jne    0x401058 <phase_3+66>
   0x0000000000401053 <+61>:    callq  0x4016a8 <explode_bomb>
   0x0000000000401058 <+66>:    mov    0xc(%rsp),%edi
   0x000000000040105c <+70>:    callq  0x400fcb <func_switch>
   0x0000000000401061 <+75>:    mov    %eax,%ebx
   0x0000000000401063 <+77>:    mov    0x8(%rsp),%edi
   0x0000000000401067 <+81>:    callq  0x400fcb <func_switch>
   0x000000000040106c <+86>:    cmp    %eax,%ebx
   0x000000000040106e <+88>:    jne    0x401076 <phase_3+96>
```

→ grow stack by 2 slots

places 0x0 12 slots away from start of rsp and 8 slots away from rsp

0 < 1 → jump to phase_3+61

·%rax stores our string

4 1

#must be equal #s

·Requirements: 2 numbers
  ·1st # in jump table - first #
  ·2nd # in jump table - second #

```
Dump of assembler code for function func_switch:
   0x0000000000400fcb <+0>:     cmp    $0x7,%edi
   0x0000000000400fce <+3>:     ja     0x400ff4 <func_switch+50>
=> 0x0000000000400fd0 <+5>:     mov    %edi,%edi
   0x0000000000400fd2 <+7>:     jmp    *0x402730(,%rdi,8)
   0x0000000000400fd9 <+14>:    mov    $0x102,%eax
   0x0000000000400fde <+19>:    ret
   0x0000000000400fdf <+20>:    mov    $0x2a4,%eax
   0x0000000000400fe4 <+25>:    ret
   0x0000000000400fe5 <+26>:    mov    $0x70,%eax
   0x0000000000400fea <+31>:    ret
   0x0000000000400feb <+32>:    mov    $0x17c,%eax
   0x0000000000400ff0 <+37>:    ret
   0x0000000000400ff1 <+38>:    mov    $0x14c,%eax
   0x0000000000400ff6 <+43>:    ret
   0x0000000000400ff7 <+44>:    mov    $0x273,%eax
   0x0000000000400ffc <+49>:    ret
   0x0000000000400ffd <+50>:    sub    $0x8,%rsp
   0x0000000000401001 <+54>:    call   0x4016a8 <explode_bomb>
   0x0000000000401006 <+59>:    mov    $0x0,%eax
   0x000000000040100b <+64>:    add    $0x8,%rsp
   0x000000000040100f <+68>:    ret
   0x0000000000401010 <+69>:    mov    $0xac,%eax
   0x0000000000401015 <+74>:    ret
```

4, 1

2

3

5

6

7

0

let X be the 1st number, where X = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}

```
Dump of assembler code for function phase_4:
   0x00000000004010be <+0>:     sub    $0x18,%rsp
   0x00000000004010c2 <+4>:     lea    0x8(%rsp),%rcx
   0x00000000004010c7 <+9>:     lea    0xc(%rsp),%rdx
   0x00000000004010cc <+14>:    mov    $0x402e59,%esi
   0x00000000004010d1 <+19>:    mov    $0x0,%eax
   0x00000000004010d6 <+24>:    call   0x400ca0 <__isoc99_sscanf@plt>
   0x00000000004010db <+29>:    cmp    $0x2,%eax
   0x00000000004010de <+32>:    jne    0x4010e7 <phase_4+41>
   0x00000000004010e0 <+34>:    cmpl   $0xe,0xc(%rsp)
   0x00000000004010e5 <+39>:    jbe    0x4010ec <phase_4+46>
   0x00000000004010e7 <+41>:    call   0x4016a8 <explode_bomb>
   0x00000000004010ec <+46>:    mov    $0xe,%edx
   0x00000000004010f1 <+51>:    mov    $0x0,%esi
   0x00000000004010f6 <+56>:    mov    0xc(%rsp),%edi
   0x00000000004010fa <+60>:    call   0x40107d <func4>
   0x00000000004010ff <+65>:    cmp    $0x1,%eax
   0x0000000000401102 <+68>:    jne    0x40110b <phase_4+77>
   0x0000000000401104 <+70>:    cmpl   $0x1,0x8(%rsp)
   0x0000000000401109 <+75>:    je     0x401110 <phase_4+82>
   0x000000000040110b <+77>:    call   0x4016a8 <explode_bomb>
   0x0000000000401110 <+82>:    add    $0x18,%rsp
   0x0000000000401114 <+86>:    ret
```

(P1)

(P2)

*second # might be 2

our first # ← Must be below or = 14 (dec)

→ copying 14 into %edx

output of first must be 1 out of func4

output of second must be 2 out of func4

```
Dump of assembler code for function func4:
   0x00000000004010 7d <+0>:     sub    $0x8,%rsp
   0x0000000000401081 <+4>:      mov    %esi,%eax
   0x0000000000401083 <+6>:      mov    %edx,%ecx
   0x0000000000401085 <+8>:      sub    %esi,%ecx
   0x0000000000401087 <+10>:     mov    %ecx,%esi
   0x0000000000401089 <+12>:     shr    $0x1f,%esi
   0x000000000040108c <+15>:     add    %ecx,%esi
   0x000000000040108e <+17>:     sar    %esi
   0x0000000000401090 <+19>:     add    %eax,%esi
   0x0000000000401092 <+21>:     cmp    %edi,%esi
   0x0000000000401094 <+23>:     jg     0x4010a2 <func4+37>
   0x0000000000401096 <+25>:     mov    $0x0,%eax
   0x000000000040109b <+30>:     jl     0x4010b0 <func4+51>
   0x000000000040109d <+32>:     add    $0x8,%rsp
   0x00000000004010a1 <+36>:     ret
   0x00000000004010a2 <+37>:     lea    -0x1(%rsi),%edx
   0x00000000004010a5 <+40>:     mov    %eax,%esi
   0x00000000004010a7 <+42>:     call   0x40107d <func4>
   0x00000000004010ac <+47>:     add    %eax,%eax
   0x00000000004010ae <+49>:     jmp    0x40109d <func4+32>
   0x00000000004010b0 <+51>:     add    $0x1,%esi
   0x00000000004010b3 <+54>:     call   0x40107d <func4>
   0x00000000004010b8 <+59>:     lea    0x1(%rax,%rax,1),%eax
   0x00000000004010bc <+63>:     jmp    0x40109d <func4+32>
```

looks at MSB. Applies logical shift

→ Decrement %rsi by 1

arg_1 = 14 %edx
arg_2 = 0 %esi
arg_3 = first number in our input. %edi

+14 move %esi (0) into %eax (return value)

+6 mov %edx (14) into %ecx

+8 %ecx = %ecx - %esi ⟶ %ecx = 14 - 0 = 14

+10: %esi = %ecx → %esi = 14

+12: %esi = %esi >>31 (logical) → %esi = 0 (signed int)

+15: %esi = %esi + %ecx ⟶ %esi = 14

+17: %esi >>0.

+19: %esi = %esi + %eax
= 14 + 0 = 14

+21: compare our first # in our input to %esi (0)

Not #6, #2, #0, #9

7-10=-3

9 1

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

7-4=3-2=1

7-4-2=1-1=0-0=0

7-3=4-5=-1-6=-7

0+

```
int y=0; (%esi)
int x=14; (%edx)
int z= our number; (%edi)

fun4 (x,y,z) {
    y = x - y;
    y = y >> 31
        (logical)
    y = y + x
    y = y >> 1; #Divide y by 2.
        (with)
    y = y + 0;
    if(z>y){
        x=x-1;
        y= y+ func4(x,y,z);
        return y;
    }
    else if (z<y){
        y=y+1
        y=func4(x,y,z) +y;
        return y;
    }
    return 0;
```

```
Dump of assembler code for function phase_5:
   0x0000000000401115 <+0>:     push   %rbx
   0x0000000000401116 <+1>:     sub    $0x10,%rsp
   0x000000000040111a <+5>:     mov    %rdi,%rbx
   0x000000000040111d <+8>:     call   0x40138a <string_length>
   0x0000000000401122 <+13>:    cmp    $0x6,%eax
   0x0000000000401125 <+16>:    jne    0x401186 <phase_5+81>
   0x0000000000401127 <+18>:    mov    $0x0,%eax
   0x000000000040112c <+23>:    movzbl (%rbx,%rax,1),%edx
   0x0000000000401130 <+27>:    and    $0xf,%edx
   0x0000000000401133 <+30>:    movzbl 0x402770(%rdx),%edx
   0x000000000040113a <+37>:    mov    %dl,0x9(%rsp,%rax,1)
   0x000000000040113e <+41>:    add    $0x1,%rax
   0x0000000000401142 <+45>:    cmp    $0x6,%rax
   0x0000000000401146 <+49>:    jne    0x40112c <phase_5+23>
   0x0000000000401148 <+51>:    movb   $0x0,0xf(%rsp)
   0x000000000040114d <+56>:    mov    $0x402780,%esi
   0x0000000000401152 <+61>:    lea    0x9(%rsp),%rdi
   0x0000000000401157 <+66>:    call   0x4013a3 <strings_not_equal>
   0x000000000040115c <+71>:    test   %eax,%eax
   0x000000000040115e <+73>:    jne    0x40116d <phase_5+88>
   0x0000000000401160 <+75>:    add    $0x10,%rsp
   0x0000000000401164 <+79>:    pop    %rbx
   0x0000000000401165 <+80>:    ret
   0x0000000000401166 <+81>:    call   0x4016a8 <explode_bomb>
   0x000000000040116b <+86>:    jmp    0x401127 <phase_5+18>
   0x000000000040116d <+88>:    call   0x4016a8 <explode_bomb>
   0x0000000000401172 <+93>:    jmp    0x401160 <phase_5+75>
End of assembler dump.
```

CP 1

<+13>: ← string length must be 6

<+18>: ← resetting return value

<+23>: %edx = %rbx + %rax = Starting address of string + 0(%rbx) (%rax) (decimal)

<+27>: %edx = %edx & 15 = 0x3b & 0xf = 1

<+30>: → %edx = 0x402770 + %rdx

<+37>: → %rsp + %rax + 9 = 1 or 'a'

loop 6 times

- chooses which elements in array.0 to select.

·Requirements·:

1) String length must be 6.

array.0 = {'m', 'a', 'd', 'u', 'i', 'e', 'r', 's', 'n', 'f', 'o', 'b', 'v', 'b', 'y', 'l'}

<+61>: → %rdi contains our first char in the strin

<+66>: → string in 0x402780 = "oilers" (Secret Stage!)

Answer: ; 4 0 5 6 7 (11) (15)

```
Dump of assembler code for function string_length:
   0x000000000040138a <+0>:     mov    $0x0,%eax
   0x000000000040138f <+5>:     cmpb   $0x0,(%rdi)
   0x0000000000401392 <+8>:     je     0x4013a2 <string_length+24>
   0x0000000000401394 <+10>:    add    $0x1,%rdi
   0x0000000000401398 <+14>:    add    $0x1,%eax
   0x000000000040139b <+17>:    cmp    $0x2710,%eax
   0x00000000004013a0 <+22>:    jne    0x40138f <string_length+5>
   0x00000000004013a2 <+24>:    ret
End of assembler dump.
```

```
Dump of assembler code for function strings_not_equal:
   0x00000000004013a3 <+0>:     push   %r12
   0x00000000004013a5 <+2>:     push   %rbp
   0x00000000004013a6 <+3>:     push   %rbx
   0x00000000004013a7 <+4>:     mov    %rdi,%rbx
   0x00000000004013aa <+7>:     mov    %rsi,%rbp
   0x00000000004013ad <+10>:    call   0x40138a <string_length>
   0x00000000004013b2 <+15>:    mov    %eax,%r12d
   0x00000000004013b5 <+18>:    mov    %rbp,%rdi
   0x00000000004013b8 <+21>:    call   0x40138a <string_length>
   0x00000000004013bd <+26>:    mov    %eax,%edx
   0x00000000004013bf <+28>:    mov    $0x1,%eax
   0x00000000004013c4 <+33>:    cmp    %edx,%r12d
   0x00000000004013c7 <+36>:    jne    0x4013fa <strings_not_equal+87>
   0x00000000004013c9 <+38>:    movzbl (%rbx),%edx
   0x00000000004013cc <+41>:    test   %dl,%dl
   0x00000000004013ce <+43>:    je     0x4013ee <strings_not_equal+75>
   0x00000000004013d0 <+45>:    mov    $0x0,%eax
   0x00000000004013d5 <+50>:    cmp    %dl,0x0(%rbp,%rax,1)
   0x00000000004013d9 <+54>:    jne    0x4013f5 <strings_not_equal+82>
   0x00000000004013db <+56>:    add    $0x1,%rax
   0x00000000004013df <+60>:    movzbl (%rbx,%rax,1),%edx
   0x00000000004013e3 <+64>:    test   %dl,%dl
   0x00000000004013e5 <+66>:    jne    0x4013d5 <strings_not_equal+50>
   0x00000000004013e7 <+68>:    mov    $0x0,%eax
   0x00000000004013ec <+73>:    jmp    0x4013fa <strings_not_equal+87>
   0x00000000004013ee <+75>:    mov    $0x0,%eax
   0x00000000004013f3 <+80>:    jmp    0x4013fa <strings_not_equal+87>
   0x00000000004013f5 <+82>:    mov    $0x1,%eax
   0x00000000004013fa <+87>:    pop    %rbx
   0x00000000004013fb <+88>:    pop    %rbp
   0x00000000004013fc <+89>:    pop    %r12
   0x00000000004013fe <+91>:    ret
End of assembler dump.
```

```
Dump of assembler code for function phase_6:
   0x0000000000401174 <+0>:     push   %r13
   0x0000000000401176 <+2>:     push   %r12
   0x0000000000401178 <+4>:     push   %rbp
   0x0000000000401179 <+5>:     push   %rbx
   0x000000000040117a <+6>:     sub    $0x58,%rsp
   0x000000000040117e <+10>:    lea    0x30(%rsp),%rsi
   0x0000000000401183 <+15>:    call   0x40192d <read_six_numbers>
   0x0000000000401188 <+20>:    lea    0x30(%rsp),%r12
   0x000000000040118d <+25>:    mov    $0x1,%r13d
   0x0000000000401193 <+31>:    jmp    0x4011bd <phase_6+73>
   0x0000000000401195 <+33>:    call   0x4016a8 <explode_bomb>
   0x0000000000401198 <+38>:    jmp    0x4011cc <phase_6+88>
   0x000000000040119c <+40>:    add    $0x1,%rbx
   0x00000000004011a0 <+44>:    cmp    $0x5,%ebx
   0x00000000004011a3 <+47>:    jg     0x4011b5 <phase_6+65>
   0x00000000004011a5 <+49>:    mov    0x30(%rsp,%rbx,4),%eax
   0x00000000004011a9 <+53>:    cmp    %eax,0x0(%rbp)
   0x00000000004011ac <+56>:    jne    0x40119c <phase_6+40>
   0x00000000004011ae <+58>:    call   0x4016a8 <explode_bomb>
   0x00000000004011b3 <+63>:    jmp    0x40119c <phase_6+40>
   0x00000000004011b5 <+65>:    add    $0x1,%r13
   0x00000000004011b9 <+69>:    add    $0x4,%r12
   0x00000000004011bd <+73>:    mov    %r12,%rbp
   0x00000000004011c0 <+76>:    mov    (%r12),%eax
   0x00000000004011c4 <+80>:    sub    $0x1,%eax
   0x00000000004011c7 <+83>:    cmp    $0x5,%eax
   0x00000000004011ca <+86>:    ja     0x401195 <phase_6+33>
   0x00000000004011cc <+88>:    cmp    $0x5,%r13d
   0x00000000004011d0 <+92>:    jg     0x4011d7 <phase_6+99>
   0x00000000004011d2 <+94>:    mov    %r13,%rbx
   0x00000000004011d5 <+97>:    jmp    0x4011a5 <phase_6+49>
   0x00000000004011d7 <+99>:    mov    $0x0,%esi
   0x00000000004011dc <+104>:   mov    0x30(%rsp,%rsi,4),%ecx
   0x00000000004011e0 <+108>:   mov    $0x1,%eax
   0x00000000004011e5 <+113>:   mov    $0x6042f0,%edx
   0x00000000004011ea <+118>:   cmp    $0x1,%ecx
   0x00000000004011ed <+121>:   jle    0x4011fa <phase_6+134>
   0x00000000004011ef <+123>:   mov    0x8(%rdx),%rdx
   0x00000000004011f3 <+127>:   add    $0x1,%eax
   0x00000000004011f6 <+130>:   cmp    %ecx,%eax
   0x00000000004011f8 <+132>:   jne    0x4011ef <phase_6+123>
   0x00000000004011fa <+134>:   mov    %rdx,(%rsp,%rsi,8)
   0x00000000004011fe <+138>:   add    $0x1,%rsi
   0x0000000000401202 <+142>:   cmp    $0x6,%rsi
   0x0000000000401206 <+146>:   jne    0x4011dc <phase_6+104>
   0x0000000000401208 <+148>:   mov    (%rsp),%rbx
   0x000000000040120c <+152>:   mov    0x8(%rsp),%rax
   0x0000000000401211 <+157>:   mov    %rax,0x8(%rbx)
   0x0000000000401215 <+161>:   mov    0x10(%rsp),%rdx
   0x000000000040121a <+166>:   mov    %rdx,0x8(%rax)
   0x000000000040121e <+170>:   mov    0x18(%rsp),%rax
   0x0000000000401223 <+175>:   mov    %rax,0x8(%rdx)
   0x0000000000401227 <+179>:   mov    0x20(%rsp),%rdx
   0x000000000040122c <+184>:   mov    %rdx,0x8(%rax)
   0x0000000000401230 <+188>:   mov    0x28(%rsp),%rax
   0x0000000000401235 <+193>:   mov    %rax,0x8(%rdx)
   0x0000000000401239 <+197>:   movq   $0x0,0x8(%rax)
   0x0000000000401241 <+205>:   mov    $0x5,%ebp
   0x0000000000401246 <+210>:   jmp    0x401251 <phase_6+221>
   0x0000000000401248 <+212>:   mov    0x8(%rbx),%rbx
   0x000000000040124c <+216>:   sub    $0x1,%ebp
--Type <RET> for more, q to quit, c to continue without paging--c
```

CP1

CP2

add 1 to the second number

%rax = %rsp + 4*%rbx + 0x30

moves to the second number.

loop

loop exits when the second number is equal to the first number.

loop exits when the second number is equal to the first number.

$r12 stores first number. $RBP too.

first number can't be greater than 5

compare if first number is 5

if >5 jump

move

examining our element again in array

loop

loop - creating linked list.

moving %1 into the second #

node 1

node 2

node 3

node 4

node 5

node 6

```
   0x0000000000401251 <+221>:   je     0x401262 <phase_6+238>
   0x0000000000401253 <+221>:   mov    0x8(%rbx),%rax
   0x0000000000401255 <+225>:   mov    (%rax),%eax
   0x0000000000401257 <+227>:   cmp    %eax,(%rbx)
   0x0000000000401259 <+229>:   jge    0x401246 <phase_6+212>
   0x000000000040125b <+231>:   call   0x4016a8 <explode_bomb>
   0x0000000000401260 <+236>:   jmp    0x401246 <phase_6+212>
   0x0000000000401262 <+238>:   cmpl   $0x21,0x2035a7(%rip)        # 0x604810 <trap>
   0x0000000000401269 <+245>:   je     0x401276 <phase_6+258>
   0x000000000040126b <+247>:   add    $0x58,%rsp
   0x000000000040126f <+251>:   pop    %rbx
   0x0000000000401270 <+252>:   pop    %rbp
   0x0000000000401271 <+253>:   pop    %r12
   0x0000000000401273 <+255>:   pop    %r13
   0x0000000000401275 <+257>:   ret
   0x0000000000401276 <+258>:   call   0x4016a8 <explode_bomb>
   0x000000000040127b <+263>:   jmp    0x40126b <phase_6+247>
```

secret phase.

- Requirements:
  1) 6 numbers
  2) First number can't be greater than 5 (has to be > 3)
  3) No two numbers are equal.

1 2 3 4 5 6

0x7fff ffff e170
0x7fff ffff e174
0x7fff ffff e178
0x7fff ffff e17c
0x7fff ffff e180
0x7fff ffff e184

where our numbers are.

X = {1, 2, 3, 4, 5, 6}

Pos
Num

loop

look for trap outside of phase 6

'\r' is the ascii value in trap

<node 1> 0x6042f0

<node 2> 0x604300

<node 3> 0x604310

<node 4> 0x604320

<node 5> 0x604330

<node 6> 0x604340

little endian

2 1 4 5 3 6

jump - jmp *0x402ff0(,%rax,8)

8(%rax) + 0x402ff0 , where %rax is the phase.

13 20 33

20 13

2 3 5 8 13 21

3 4 7 11 18 24

$$\begin{array}{r} 11 \\ +18 \\ \hline 29 \end{array}$$

4 5 9 14 23 37

$$\begin{array}{r} 23 \\ +14 \\ \hline 37 \end{array}$$

2 4 8 12 20 32

1 4 5 9 14

2 5 7 12 21 33

2 5 7 12 19

3 5 8 13 21 34

Secret_phase_2 1 6 7 13 20 33

1 5

```
Dump of assembler code for function explode_bomb:
   0x00000000004016a8 <+0>:     sub    $0xd8,%rsp
   0x00000000004016af <+7>:     cmpl   $0x6,0x20315e(%rip)        # 0x604814 <num_input_strings>
   0x00000000004016b6 <+14>:    ja     0x4016cf <explode_bomb+39>
   0x00000000004016b8 <+16>:    mov    0x203156(%rip),%eax        # 0x604814 <num_input_strings>
   0x00000000004016be <+22>:    jmp    *0x402ff0(,%rax,8)
   0x00000000004016c5 <+29>:    mov    $0x402dd3,%edi
   0x00000000004016ca <+34>:    call   0x400bb0 <puts@plt>
   0x00000000004016cf <+39>:    movabs $0x6d20756f79206f53,%rax
   0x00000000004016d9 <+49>:    movabs $0x7420746920656461,%rdx
   0x00000000004016e3 <+59>:    mov    %rax,(%rsp)
   0x00000000004016e7 <+63>:    mov    %rdx,0x8(%rsp)
   0x00000000004016ec <+68>:    movabs $0x337320656874206f,%rax
   0x00000000004016f6 <+78>:    movabs $0x6168702074657263,%rdx
   0x0000000000401700 <+88>:    mov    %rax,0x10(%rsp)
   0x0000000000401705 <+93>:    mov    %rdx,0x18(%rsp)
   0x000000000040170a <+98>:    movabs $0x63694e20202e6573,%rax
   0x0000000000401714 <+108>:   movabs $0x640a7475622202c65,%rdx
   0x000000000040171e <+118>:   mov    %rax,0x20(%rsp)
   0x0000000000401723 <+123>:   mov    %rdx,0x28(%rsp)
   0x0000000000401728 <+128>:   movabs $0x6c65742074276e6f,%rax
   0x0000000000401732 <+138>:   movabs $0x656e6f796e61206c,%rdx
   0x000000000040173c <+148>:   mov    %rax,0x30(%rsp)
   0x0000000000401741 <+153>:   mov    %rdx,0x38(%rsp)
   0x0000000000401746 <+158>:   movabs $0x69207475 6f626120,%rax
   0x0000000000401750 <+168>:   movabs $0x6365707365282074,%rdx
   0x000000000040175a <+178>:   mov    %rax,0x40(%rsp)
   0x000000000040175f <+183>:   mov    %rdx,0x48(%rsp)
   0x0000000000401764 <+188>:   movabs $0x6e6f20796c6c6169,%rax
   0x000000000040176e <+198>:   movabs $0x29617a7a61695020,%rdx
   0x0000000000401778 <+208>:   mov    %rax,0x50(%rsp)
   0x000000000040177d <+213>:   mov    %rdx,0x58(%rsp)
   0x0000000000401782 <+218>:   movabs $0x756f792066490a2e,%rax
   0x000000000040178c <+228>:   movabs $0x20756f79206f6420,%rdx
   0x0000000000401796 <+238>:   mov    %rax,0x60(%rsp)
   0x000000000040179b <+243>:   mov    %rdx,0x68(%rsp)
   0x00000000004017a0 <+248>:   movabs $0x65672074276e6f77,%rax
   0x00000000004017aa <+258>:   movabs $0x786520796e612074,%rdx
   0x00000000004017b4 <+268>:   mov    %rax,0x70(%rsp)
   0x00000000004017b9 <+273>:   mov    %rdx,0x78(%rsp)
   0x00000000004017be <+278>:   movabs $0x6465726320617274,%rax
   0x00000000004017c8 <+288>:   movabs $0x77796e6120217469,%rdx
   0x00000000004017d2 <+298>:   mov    %rax,0x80(%rsp)
   0x00000000004017da <+306>:   mov    %rdx,0x88(%rsp)
   0x00000000004017e2 <+314>:   movabs $0x6e69660a2c737961,%rax
   0x00000000004017ec <+324>:   movabs $0x20746920676e6964,%rdx
   0x00000000004017f6 <+334>:   mov    %rax,0x90(%rsp)
   0x00000000004017fe <+342>:   mov    %rdx,0x98(%rsp)
   0x0000000000401806 <+350>:   movabs $0x766c6f7320646e61,%rax
   0x0000000000401810 <+360>:   movabs $0x6120746920676e69,%rdx
   0x000000000040181a <+370>:   mov    %rax,0xa0(%rsp)
   0x0000000000401822 <+378>:   mov    %rdx,0xa8(%rsp)
   0x000000000040182a <+386>:   movabs $0x6574697571206572,%rax
   0x0000000000401834 <+396>:   movabs $0x6572656666696420,%rdx
```

```
   0x0000000000401834 <+396>:   movabs $0x6572656666696420,%rdx
--Type <RET> for more, q to quit, c to continue without paging--c
   0x000000000040183e <+406>:   mov    %rax,0xb0(%rsp)
   0x0000000000401846 <+414>:   mov    %rdx,0xb8(%rsp)
   0x000000000040184e <+422>:   movabs $0x2e2e2e746e,%rax
   0x0000000000401858 <+432>:   mov    %rax,0xc0(%rsp)
   0x0000000000401860 <+440>:   cmpl   $0x6,0x202fad(%rip)        # 0x604814 <num_input_strings>
   0x0000000000401867 <+447>:   jne    0x401376 <explode_bomb+462>
   0x0000000000401869 <+449>:   cmpl   $0x21,0x202fa0(%rip)       # 0x604810 <trap>
   0x0000000000401870 <+456>:   je     0x401376 <explode_bomb+602>
   0x0000000000401876 <+462>:   mov    $0x402e2d,%edi
   0x000000000040187b <+467>:   call   0x400bb0 <puts@plt>
   0x0000000000401880 <+472>:   mov    $0x402e36,%edi
   0x0000000000401885 <+477>:   call   0x400bb0 <puts@plt>
   0x000000000040188a <+482>:   mov    $0x0,%edi
   0x000000000040188f <+487>:   call   0x4015e0 <send_msg>* don't run this!
   0x0000000000401894 <+492>:   mov    $0x402d58,%edi
   0x0000000000401899 <+497>:   call   0x400bb0 <puts@plt>
   0x000000000040189e <+502>:   mov    $0x8,%edi
   0x00000000004018a3 <+507>:   call   0x400c10 <exit@plt>
   0x00000000004018a8 <+512>:   mov    $0x402d30,%edi
   0x00000000004018ad <+517>:   call   0x400bb0 <puts@plt>
   0x00000000004018b2 <+522>:   jmp    0x4016cf <explode_bomb+39>
   0x00000000004018b7 <+527>:   mov    $0x402dd8,%edi
   0x00000000004018bc <+532>:   call   0x400bb0 <puts@plt>
   0x00000000004018c1 <+537>:   jmp    0x4016cf <explode_bomb+39>
   0x00000000004018c6 <+542>:   mov    $0x402ded,%edi
   0x00000000004018cb <+547>:   call   0x400bb0 <puts@plt>
   0x00000000004018d0 <+552>:   jmp    0x4016cf <explode_bomb+39>
   0x00000000004018d5 <+557>:   mov    $0x402e03,%edi
   0x00000000004018da <+562>:   call   0x400bb0 <puts@plt>
   0x00000000004018df <+567>:   jmp    0x4016cf <explode_bomb+39>
   0x00000000004018e4 <+572>:   mov    $0x402e1a,%edi
   0x00000000004018e9 <+577>:   call   0x400bb0 <puts@plt>
   0x00000000004018ee <+582>:   jmp    0x4016cf <explode_bomb+39>
   0x00000000004018f3 <+587>:   mov    $0x402e23,%edi
   0x00000000004018f8 <+592>:   call   0x400bb0 <puts@plt>
   0x00000000004018fd <+597>:   jmp    0x4016cf <explode_bomb+39>
   0x0000000000401902 <+602>:   mov    %rsp,%rdi
   0x0000000000401905 <+605>:   call   0x400bb0 <puts@plt>
   0x0000000000401905 <+610>:   call   0x40166d <phase_defused>
   0x000000000040190f <+615>:   movl   $0x0,0x202ef7(%rip)        # 0x604810 <trap>
   0x0000000000401919 <+625>:   mov    $0x0,%eax
   0x000000000040191e <+630>:   call   0x4012ba <s3cret_phase>
   0x0000000000401923 <+635>:   mov    $0x0,%edi
   0x0000000000401928 <+640>:   call   0x400c10 <exit@plt>
```

```
(gdb) x /40bs 0x202fad+0x401860
0x60480d <scratch+77>:  ""
0x60480e <scratch+78>:  ""
0x60480f <scratch+79>:  ""
0x604810 <trap>:        "\r"
0x604812 <trap+2>:      ""
0x604813 <trap+3>:      ""
0x604814 <num_input_strings>:   "\004"
0x604816 <num_input_strings+2>: ""
0x604817 <num_input_strings+3>: ""
0x604818:       ""
0x604819:       ""
0x60481a:       ""
0x60481b:       ""
0x60481c:       ""
0x60481d:       ""
0x60481e:       ""
0x60481f:       ""
0x604820 <input_strings>:       "h"
0x604822 <input_strings+2>:     ""
0x604823 <input_strings+3>:     ""
0x604824 <input_strings+4>:     ""
0x604825 <input_strings+5>:     ""
0x604826 <input_strings+6>:     ""
0x604827 <input_strings+7>:     ""
0x604828 <input_strings+8>:     ""
0x604829 <input_strings+9>:     ""
0x60482a <input_strings+10>:    ""
0x60482b <input_strings+11>:    ""
0x60482c <input_strings+12>:    ""
0x60482d <input_strings+13>:    ""
0x60482e <input_strings+14>:    ""
0x60482f <input_strings+15>:    ""
0x604830 <input_strings+16>:    ""
0x604831 <input_strings+17>:    ""
0x604832 <input_strings+18>:    ""
0x604833 <input_strings+19>:    ""
0x604834 <input_strings+20>:    ""
0x604835 <input_strings+21>:    ""
0x604836 <input_strings+22>:    ""
0x604837 <input_strings+23>:    ""
```

```
(gdb) x /150d 0x202fad+0x401860
0x60480d <scratch+77>:  0       0       0       13      0       0       0       4
0x604815 <num_input_strings+1>: 0       0       0       0       0       0       0       0
0x60481d:       0       0       0       104     0       0       0       0
0x604825 <input_strings+5>:     0       0       0       0       0       0       0       0
0x60482d <input_strings+13>:    0       0       0       0       0       0       0       0
0x604835 <input_strings+21>:    0       0       0       0       0       0       0       0
0x60483d <input_strings+29>:    0       0       0       0       0       0       0       0
0x604845 <input_strings+37>:    0       0       0       0       0       0       0       0
0x60484d <input_strings+45>:    0       0       0       0       0       0       0       0
0x604855 <input_strings+53>:    0       0       0       0       0       0       0       0
0x60485d <input_strings+61>:    0       0       0       0       0       0       0       0
0x604865 <input_strings+69>:    0       0       0       0       0       0       0       0
0x60486d <input_strings+77>:    0       0       0       49      32      50      32      51
0x604875 <input_strings+85>:    32      53      32      56      32      49      51      0
0x60487d <input_strings+93>:    0       0       0       0       0       0       0       0
0x604885 <input_strings+101>:   0       0       0       0       0       0       0       0
0x60488d <input_strings+109>:   0       0       0       0       0       0       0       0
0x604895 <input_strings+117>:   0       0       0       0       0       0       0       0
0x60489d <input_strings+125>:   0       0       0       0       0       0
```

* phase_2 <+45> mov 3 into 0x604810 <trap>

trap contains our last integer in the six numbers

* last number in phase_2 must contain 37.

```
Dump of assembler code for function s3cret_phase:
   0x0000000000401200 <+0>:     push   %rbx
   0x0000000000401201 <+1>:     call   0x4019c0 <read_line>
   0x0000000000401206 <+6>:     mov    %rax,%rdi
   0x0000000000401209 <+9>:     mov    $0xa,%edx
   0x000000000040120e <+14>:    mov    $0x0,%esi
   0x0000000000401213 <+19>:    call   0x400e80 <strtol@plt>
   0x0000000000401218 <+24>:    mov    %rax,%rbx
   0x000000000040121b <+27>:    lea    -0x1(%rax),%eax
   0x000000000040121e <+30>:    cmp    $0x3e8,%eax
   0x0000000000401223 <+35>:    ja     0x401300 <s3cret_phase+81>
   0x0000000000401229 <+37>:    mov    %ebx,%esi
   0x000000000040122b <+39>:    mov    $0x604110,%edi
   0x0000000000401230 <+44>:    call   0x401274 <fun7>
   0x0000000000401235 <+49>:    cmp    $0x4,%eax
   0x0000000000401238 <+52>:    jne    0x401312 <s3cret_phase+88>
   0x000000000040123a <+54>:    mov    $0x402788,%edi
   0x0000000000401240 <+59>:    call   0x4006b0 <puts@plt>
   0x0000000000401245 <+64>:    mov    $0x4027b0,%edi
   0x000000000040124a <+69>:    call   0x4006b0 <puts@plt>
   0x000000000040124f <+74>:    call   0x401664 <phase_defused>
   0x0000000000401254 <+79>:    pop    %rbx
   0x0000000000401255 <+80>:    ret
   0x0000000000401256 <+81>:    call   0x4016a8 <explode_bomb>
   0x000000000040125b <+86>:    jmp    0x401229 <s3cret_phase+37>
   0x000000000040125d <+88>:    call   0x4016a8 <explode_bomb>
   0x0000000000401262 <+93>:    jmp    0x4012f0 <s3cret_phase+54>
```

- %rax and %rdi contains our number string "22"
- %edi contains 36

- Requirements:
  - 1) # must be less than 1000

```
Dump of assembler code for function fun7:
=> 0x000000000040127d <+0>:     test   %rdi,%rdi
   0x0000000000401280 <+3>:     je     0x4012b4 <fun7+55>
   0x0000000000401282 <+5>:     sub    $0x8,%rsp
   0x0000000000401286 <+9>:     mov    (%rdi),%edx
   0x0000000000401288 <+11>:    cmp    %esi,%edx
   0x000000000040128a <+13>:    jg     0x401298 <fun7+27>
   0x000000000040128c <+15>:    mov    $0x0,%eax
   0x0000000000401291 <+20>:    jne    0x4012a5 <fun7+40>
   0x0000000000401293 <+22>:    add    $0x8,%rsp
   0x0000000000401297 <+26>:    ret
   0x0000000000401298 <+27>:    mov    0x8(%rdi),%rdi
   0x000000000040129c <+31>:    call   0x40127d <fun7>
   0x00000000004012a1 <+36>:    add    %eax,%eax
   0x00000000004012a3 <+38>:    jmp    0x401293 <fun7+22>
   0x00000000004012a5 <+40>:    mov    0x10(%rdi),%rdi
   0x00000000004012a9 <+44>:    call   0x40127d <fun7>
   0x00000000004012ae <+49>:    lea    0x1(%rax,%rax,1),%eax
   0x00000000004012b2 <+53>:    jmp    0x401293 <fun7+22>
   0x00000000004012b4 <+55>:    mov    $0xffffffff,%eax
   0x00000000004012b9 <+60>:    ret
End of assembler dump.
```

- 1) %rax + %rax
- 2) %rax + %rax + 1

#7