CSAPPLab2 BombLab

这是一个关于程序的机器级表示的实验，重点在于熟悉汇编代码，掌握常用的汇编指令，读懂程序的运作流程。该实验又被称为“二进制炸弹”实验，共有六个“*phase*”，对于每个“*phase*”，需要正确输入一段字符串，保证代码中的函数“*explode_bomb*”不被执行，否则程序终止并打印出“***BOOM!!!***”。

实验材料中给了 c 文件“*bomb.c*”和二进制可执行文件“*bomb*”。首先应使用“*objdump*”命令反汇编“*bomb*”文件，将得到的汇编代码存入 txt 文件中，便于查看；然后采用“*gdb*”进行程序调试，找到应当输入的字符串。

```
linux > objdump -d bomb > obj.txt
```

实验开始之前，我们应先了解一个 *x86-64* 的中央处理单元（*CPU*）包含一组 16 个存储 64 位值的**通用目的寄存器**，如下图所示：

Figure 3.2 **Integer registers.** The low-order portions of all 16 registers can be accessed as byte, word (16-bit), double word (32-bit), and quad word (64-bit) quantities.

这些寄存器用来存储整数数据和指针。特别地，$\%rax$ 可作为调用函数的返回值；$\%rdi$, $\%rsi$, $\%rdx$, $\%rcx$, $\%r8$ 和 $\%r9$ 分别可作为调用函数的第 $1 \sim 6$ 个输入参数。

phase_1

```
0000000000400ee0 <phase_1>:
  400ee0:   48 83 ec 08             sub    $0x8,%rsp
  400ee4:   be 00 24 40 00          mov    $0x402400,%esi
  400ee9:   e8 4a 04 00 00          callq  401338 <strings_not_equal>
  400eee:   85 c0                   test   %eax,%eax
  400ef0:   74 05                   je     400ef7 <phase_1+0x17>
  400ef2:   e8 43 05 00 00          callq  40143a <explode_bomb>
  400ef7:   48 83 c4 08             add    $0x8,%rsp
  400efb:   c3                      retq
```

本题目先尝试输入任意字符串，例如 "*a*"，进行调试。程序运行至断点 *0x400ee0* 处，观察寄存器 *%rax* , *%rdi* 中的值均为 *0x603780* 。猜想寄存器 *%rax* 保存着输入函数的返回值，返回值为输入字符串的首地址，用如下命令输出其值，发现符合猜想。

```
(gdb) p (char*) $rax
$1 = 0x603780 <input_strings> "a"
```

程序逐步运行至 *0x400ee4* 处，将值 *0x402400* 赋值给 *%esi*，下一步即调用 *strings_not_equal* 函数，鉴于调用函数时，寄存器 *%rdi* 和 *%rsi* 分别作为函数的第一和第二个参数，且此时 *%rdi* 中存储着输入字符串的首地址，猜想 *%rsi* 中同样存储着一段字符串的首地址，代入 *strings_not_equal* 函数进行比较。程序地址 *0x400eee* 处执行指令 `test %eax,%eax`，此语句功能为判断寄存器 *%eax* 的值是否为 0，接下来可以看到若值为 0，则跳转到程序地址 *0x400ef7* 处，程序顺利返回；否则程序会调用 *explode_bomb* 函数，输出 *BOOM!!!* 终止。采用如下命令打印出地址 *0x402400* 的值，发现是一段字符串。

```
(gdb) p (char *) 0x402400
$6 = 0x402400 "Border relations with Canada have never been better."
```

重新执行函数，将此字符串输入，成功运行！

---

phase_2

```
0000000000400efc <phase_2>:
  400efc:   55                      push   %rbp
  400efd:   53                      push   %rbx
  400efe:   48 83 ec 28             sub    $0x28,%rsp
  400f02:   48 89 e6                mov    %rsp,%rsi
  400f05:   e8 52 05 00 00          callq  40145c <read_six_numbers>
  400f0a:   83 3c 24 01             cmpl   $0x1,(%rsp)
  400f0e:   74 20                   je     400f30 <phase_2+0x34>
  400f10:   e8 25 05 00 00          callq  40143a <explode_bomb>
  400f15:   eb 19                   jmp    400f30 <phase_2+0x34>
  400f17:   8b 43 fc                mov    -0x4(%rbx),%eax
  400f1a:   01 c0                   add    %eax,%eax
  400f1c:   39 03                   cmp    %eax,(%rbx)
  400f1e:   74 05                   je     400f25 <phase_2+0x29>
  400f20:   e8 15 05 00 00          callq  40143a <explode_bomb>
  400f25:   48 83 c3 04             add    $0x4,%rbx
  400f29:   48 39 eb                cmp    %rbp,%rbx
  400f2c:   75 e9                   jne    400f17 <phase_2+0x1b>
  400f2e:   eb 0c                   jmp    400f3c <phase_2+0x40>
  400f30:   48 8d 5c 24 04          lea    0x4(%rsp),%rbx
  400f35:   48 8d 6c 24 18          lea    0x18(%rsp),%rbp
  400f3a:   eb db                   jmp    400f17 <phase_2+0x1b>
  400f3c:   48 83 c4 28             add    $0x28,%rsp
  400f40:   5b                      pop    %rbx
  400f41:   5d                      pop    %rbp
  400f42:   c3                      retq
```

观察该代码段开始部分调用 *read_six_numbers* 函数，可知应输入包含六个数字的字符串（这也是试过常规字符串出错之后才知道的），本次输入 *“1,2,3,4,5,6”* 进行调试。程序运行至断点 *0x400efc* 处，观察此时各寄存器存储的值（主要是该函数中使用的寄存器）：

REG	VAL		REG	VAL
%rax	0x6037d0		%rbx	0x0
%rsi	0x6037d0		%rdi	0x6037d0
%rbp	0x402210		%rsp	0x7fff-ffff-def8

寄存器 *%rax* 和 *%rdi* 均存储着输入字符串的首地址 *0x6037d0*，可输入如下命令查看：

```
(gdb) p (char *) 0x6037d0
$4 = 0x6037d0 <input_strings+80> "1 2 3 4 5 6"
```

```
400efc: 55                   push   %rbp
400efd: 53                   push   %rbx
400efe: 48 83 ec 28          sub    $0x28,%rsp
400f02: 48 89 e6             mov    %rsp,%rsi
400f05: e8 52 05 00 00       callq  40145c <read_six_numbers>
```

继续执行至程序地址 *0x400f05*，此时 *%rsp* 中值为 *0x7fff-ffff-dec0*，将调用函数 *read_six_numbers* 读取输入字符串中的六个数字，并将其保存到运行栈中如下位置：

Val		%rsp
??		0x7fff-ffff-def8
rbp		0x7fff-ffff-def0
rbx		0x7fff-ffff-dee8
??		0x7fff-ffff-dee0
??		0x7fff-ffff-ded8
6	5	0x7fff-ffff-ded0
4	3	0x7fff-ffff-dec8
2	1	0x7fff-ffff-dec0

```
400f0a: 83 3c 24 01          cmpl   $0x1,(%rsp)
400f0e: 74 20                je     400f30 <phase_2+0x34>
400f10: e8 25 05 00 00       callq  40143a <explode_bomb>
```

此时 *%rsp* 中值仍为 *0x7fff-ffff-dec0*，取其存储的值与 *0x1* 比较，若相等，调程序跳转到 *0x400f30*，否则会执行函数 *explode_bomb*。所以，**可判断出第一个数应为 *1***。

```
400f30: 48 8d 5c 24 04       lea    0x4(%rsp),%rbx
400f35: 48 8d 6c 24 18       lea    0x18(%rsp),%rbp
400f3a: eb db                jmp    400f17 <phase_2+0x1b>
```

程序跳转到 *0x400f30*，将 *%rbx* 赋值为指向第二个输入值的地址 *0x7fff-ffff-dec4*，将 *%rbp* 的值赋值为指向输入最后一个值的下一个地址 *0x7fff-ffff-ded8*，跳转到 *0x400f17*。

```
400f17: 8b 43 fc             mov    -0x4(%rbx),%eax
400f1a: 01 c0                add    %eax,%eax
400f1c: 39 03                cmp    %eax,(%rbx)
400f1e: 74 05                je     400f25 <phase_2+0x29>
400f20: e8 15 05 00 00       callq  40143a <explode_bomb>
400f25: 48 83 c3 04          add    $0x4,%rbx
400f29: 48 39 eb             cmp    %rbp,%rbx
400f2c: 75 e9                jne    400f17 <phase_2+0x1b>
400f2e: eb 0c                jmp    400f3c <phase_2+0x40>
```

程序执行至 *0x400f17* 处，将 *%rbx* 指向元素的前一个值赋给 *%eax* ，并将 *%eax* 的值乘以 *2* 与 *%rbx* 指向的值作比较，如果相等，则程序跳转至 *0x400f25* ，否则执行函数 *explode_bomb* 。所以**可判断第二个值为第一个值的 *2* 倍，即为 *2* 。**程序地址 *0x400f25* 处，将 *%rbx* 的值指向下一个元素，并与 *%rbp* 的值相比较，判断是否指向了最后一个元素末尾，若两值不相等，则程序再次跳转至 *0x400f17* 重复上一次的步骤。由此可知，**输入数值大小应为前一个值的 *2* 倍 。**

```
400f3c: 48 83 c4 28             add    $0x28,%rsp
400f40: 5b                      pop    %rbx
400f41: 5d                      pop    %rbp
400f42: c3                      retq
```

上步遍历完所有数据后，程序跳转至 *0x400f3c* ，运行结束。所以，正确输入的字符串应为：***“1,2,4,8,16,32”*** 。

---

phase_3

```
0000000000400f43 <phase_3>:
  400f43:   48 83 ec 18             sub    $0x18,%rsp
  400f47:   48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
  400f4c:   48 8d 54 24 08          lea    0x8(%rsp),%rdx
  400f51:   be cf 25 40 00          mov    $0x4025cf,%esi
  400f56:   b8 00 00 00 00          mov    $0x0,%eax
  400f5b:   e8 90 fc ff ff          callq  400bf0 <__isoc99_sscanf@plt>
  400f60:   83 f8 01                cmp    $0x1,%eax
  400f63:   7f 05                   jg     400f6a <phase_3+0x27>
  400f65:   e8 d0 04 00 00          callq  40143a <explode_bomb>
  400f6a:   83 7c 24 08 07          cmpl   $0x7,0x8(%rsp)
  400f6f:   77 3c                   ja     400fad <phase_3+0x6a>
  400f71:   8b 44 24 08             mov    0x8(%rsp),%eax
  400f75:   ff 24 c5 70 24 40 00    jmpq   *0x402470(,%rax,8)
  400f7c:   b8 cf 00 00 00          mov    $0xcf,%eax
  400f81:   eb 3b                   jmp    400fbe <phase_3+0x7b>
  400f83:   b8 c3 02 00 00          mov    $0x2c3,%eax
  400f88:   eb 34                   jmp    400fbe <phase_3+0x7b>
  400f8a:   b8 00 01 00 00          mov    $0x100,%eax
  400f8f:   eb 2d                   jmp    400fbe <phase_3+0x7b>
  400f91:   b8 85 01 00 00          mov    $0x185,%eax
  400f96:   eb 26                   jmp    400fbe <phase_3+0x7b>
  400f98:   b8 ce 00 00 00          mov    $0xce,%eax
  400f9d:   eb 1f                   jmp    400fbe <phase_3+0x7b>
  400f9f:   b8 aa 02 00 00          mov    $0x2aa,%eax
  400fa4:   eb 18                   jmp    400fbe <phase_3+0x7b>
  400fa6:   b8 47 01 00 00          mov    $0x147,%eax
  400fab:   eb 11                   jmp    400fbe <phase_3+0x7b>
  400fad:   e8 88 04 00 00          callq  40143a <explode_bomb>
  400fb2:   b8 00 00 00 00          mov    $0x0,%eax
  400fb7:   eb 05                   jmp    400fbe <phase_3+0x7b>
  400fb9:   b8 37 01 00 00          mov    $0x137,%eax
  400fbe:   3b 44 24 0c             cmp    0xc(%rsp),%eax
  400fc2:   74 05                   je     400fc9 <phase_3+0x86>
  400fc4:   e8 71 04 00 00          callq  40143a <explode_bomb>
  400fc9:   48 83 c4 18             add    $0x18,%rsp
  400fcd:   c3                      retq
```

本题目应输入包含两个数字的字符串（试错后得知）。

```
400f43: 48 83 ec 18             sub    $0x18,%rsp
400f47: 48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
400f4c: 48 8d 54 24 08          lea    0x8(%rsp),%rdx
400f51: be cf 25 40 00          mov    $0x4025cf,%esi
400f56: b8 00 00 00 00          mov    $0x0,%eax
400f5b: e8 90 fc ff ff          callq  400bf0 <__isoc99_sscanf@plt>
400f60: 83 f8 01                cmp    $0x1,%eax
400f63: 7f 05                   jg     400f6a <phase_3+0x27>
400f65: e8 d0 04 00 00          callq  40143a <explode_bomb>
```

本次输入字符串 *"1,2"* 进行调试，程序运行至地址 *0x400f5b* 处（未调用函数）时部分寄存器值如下所示：

REG	VAL		REG	VAL
%rax	0x0		%rbx	0x0
%rcx	0x7fff-ffff-deec		%rdx	0x7fff-ffff-dee8
%rsi	0x4025cf		%rdi	0x603820
%rsp	0x7fff-ffff-dee0			

调用函数前向寄存器 *%rdx* 和 *%rcx* 赋值，很可能其为函数输入参数的一部分，可初步判断程序地址 *0x400f5b* 调用的函数包含四个输入参数，第一个输入参数 *%rdi* 指向输入字符串的首地址，第二个参数 *%rsi* 同样指向一个地址，我们下边打印出它的值，第三，四个参数都指向运行时栈上的地址（我们下边可以得知是将输入的两个数字存储在 *%rcx* 和 *%rdx* 指向的位置）。

```
(gdb) p (char *) $rdi
$3 = 0x603820 <input_strings+160> "1 2"
(gdb) p (char *) 0x4025cf
$4 = 0x4025cf "%d %d"
```

调用函数后，函数返回值保存至寄存器 *%rax* 中，为 *2* 。此时 *%rsp* 的值仍为 *0x7fff-ffff-dee0* ，读取的输入的两个值分别保存在栈地址 *0x7fff-ffff-dee8* 和 *0x7fff-ffff-deec* 处，打印输出如下：

```
(gdb) p /x *(int *) 0x7fffffffdee8
$6 = 0x1
(gdb) p /x *(int *) 0x7fffffffdeec
$7 = 0x2
```

程序运行至地址 *0x400f60* 处执行语句 `cmp $0x1,%eax` ，*%eax* 的值大于 *1* 时，程序跳转至 *0x400f6a* ，否则执行函数 *explode_bomb* 。所以，**输入字符串至少应为两个数字。**

```
400f6a: 83 7c 24 08 07          cmpl   $0x7,0x8(%rsp)
400f6f: 77 3c                   ja     400fad <phase_3+0x6a>
400f71: 8b 44 24 08             mov    0x8(%rsp),%eax
400f75: ff 24 c5 70 24 40 00    jmpq   *0x402470(,%rax,8)
400f7c: b8 cf 00 00 00          mov    $0xcf,%eax
400f81: eb 3b                   jmp    400fbe <phase_3+0x7b>
400f83: b8 c3 02 00 00          mov    $0x2c3,%eax
400f88: eb 34                   jmp    400fbe <phase_3+0x7b>
400f8a: b8 00 01 00 00          mov    $0x100,%eax
400f8f: eb 2d                   jmp    400fbe <phase_3+0x7b>
400f91: b8 85 01 00 00          mov    $0x185,%eax
400f96: eb 26                   jmp    400fbe <phase_3+0x7b>
```

```
400f98: b8 ce 00 00 00          mov    $0xce,%eax
400f9d: eb 1f                   jmp    400fbe <phase_3+0x7b>
400f9f: b8 aa 02 00 00          mov    $0x2aa,%eax
400fa4: eb 18                   jmp    400fbe <phase_3+0x7b>
400fa6: b8 47 01 00 00          mov    $0x147,%eax
400fab: eb 11                   jmp    400fbe <phase_3+0x7b>
400fad: e8 88 04 00 00          callq  40143a <explode_bomb>
400fb2: b8 00 00 00 00          mov    $0x0,%eax
400fb7: eb 05                   jmp    400fbe <phase_3+0x7b>
400fb9: b8 37 01 00 00          mov    $0x137,%eax
400fbe: 3b 44 24 0c             cmp    0xc(%rsp),%eax
400fc2: 74 05                   je     400fc9 <phase_3+0x86>
400fc4: e8 71 04 00 00          callq  40143a <explode_bomb>
400fc9: 48 83 c4 18             add    $0x18,%rsp
400fcd: c3                      retq
```

程序运行至地址 *0x400f6a* 处，将输入的第一个数与数值 *7* 相比较，若输入值大于 *7* ，则跳转到地址 *0x400fad* ，执行 *explode_bomb* 函数。所以，**输入的第一个值应满足小于等于 *7* 。**

接下来将输入的第一个值赋值给 *%eax* ，并跳转到以 *0x402470* 为基地址，*%rax∗8* 为偏移量的地址处保存的值

。因 *%rax* 最大值为 *7* ，所以最大地址为 *0x4024a8* 。然后根据根据各地址对应的值跳转到程序相应的位置，将一个定值赋值给 *%eax* 。最后程序跳转到地址 *0x400fbe* 处，将输入的第二个值与 *%rax* 比较，若两值相等，则正确运行结束，否则会执行函数 *explode_bomb* 。各地址对应的值及相应第二个输入应为某值如下图所示：

%rax	Add	Val	%eax	
0	0x402470	0x400f7c	0xcf	207
1	0x402478	0x400fb9	0x137	311
2	0x402480	0x400f83	0x2c3	707
3	0x402488	0x400f8a	0x100	256
4	0x402490	0x400f91	0x185	389
5	0x402498	0x400f98	0xce	206
6	0x4024a0	0x400f9f	0x2aa	682
7	0x4024a8	0x400fa6	0x147	327

所以，**输入值应为上图中第一列值和最后一列值的对应组合。**

---

phase_4

```
0000000000400fce <func4>:
  400fce:   48 83 ec 08             sub    $0x8,%rsp
  400fd2:   89 d0                   mov    %edx,%eax
  400fd4:   29 f0                   sub    %esi,%eax
  400fd6:   89 c1                   mov    %eax,%ecx
  400fd8:   c1 e9 1f                shr    $0x1f,%ecx
  400fdb:   01 c8                   add    %ecx,%eax
  400fdd:   d1 f8                   sar    %eax
  400fdf:   8d 0c 30                lea    (%rax,%rsi,1),%ecx
  400fe2:   39 f9                   cmp    %edi,%ecx
  400fe4:   7e 0c                   jle    400ff2 <func4+0x24>
  400fe6:   8d 51 ff                lea    -0x1(%rcx),%edx
  400fe9:   e8 e0 ff ff ff          callq  400fce <func4>
  400fee:   01 c0                   add    %eax,%eax
  400ff0:   eb 15                   jmp    401007 <func4+0x39>
```

```
  400ff2:   b8 00 00 00 00          mov    $0x0,%eax
  400ff7:   39 f9                   cmp    %edi,%ecx
  400ff9:   7d 0c                   jge    401007 <func4+0x39>
  400ffb:   8d 71 01                lea    0x1(%rcx),%esi
  400ffe:   e8 cb ff ff ff          callq  400fce <func4>
  401003:   8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
  401007:   48 83 c4 08             add    $0x8,%rsp
  40100b:   c3                      retq

000000000040100c <phase_4>:
  40100c:   48 83 ec 18             sub    $0x18,%rsp
  401010:   48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
  401015:   48 8d 54 24 08          lea    0x8(%rsp),%rdx
  40101a:   be cf 25 40 00          mov    $0x4025cf,%esi
  40101f:   b8 00 00 00 00          mov    $0x0,%eax
  401024:   e8 c7 fb ff ff          callq  400bf0 <__isoc99_sscanf@plt>
  401029:   83 f8 02                cmp    $0x2,%eax
  40102c:   75 07                   jne    401035 <phase_4+0x29>
  40102e:   83 7c 24 08 0e          cmpl   $0xe,0x8(%rsp)
  401033:   76 05                   jbe    40103a <phase_4+0x2e>
  401035:   e8 00 04 00 00          callq  40143a <explode_bomb>
  40103a:   ba 0e 00 00 00          mov    $0xe,%edx
  40103f:   be 00 00 00 00          mov    $0x0,%esi
  401044:   8b 7c 24 08             mov    0x8(%rsp),%edi
  401048:   e8 81 ff ff ff          callq  400fce <func4>
  40104d:   85 c0                   test   %eax,%eax
  40104f:   75 07                   jne    401058 <phase_4+0x4c>
  401051:   83 7c 24 0c 00          cmpl   $0x0,0xc(%rsp)
  401056:   74 05                   je     40105d <phase_4+0x51>
  401058:   e8 dd 03 00 00          callq  40143a <explode_bomb>
  40105d:   48 83 c4 18             add    $0x18,%rsp
  401061:   c3                      retq
```

本题目应输入包含两个数字的字符串（从 *phase_4* 的前几行代码可以看出和 *phase_3* 基本一致），且输入字符串仍为 *"1,2"* 进行调试。

> ```
> 40100c: 48 83 ec 18             sub    $0x18,%rsp
> 401010: 48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
> 401015: 48 8d 54 24 08          lea    0x8(%rsp),%rdx
> 40101a: be cf 25 40 00          mov    $0x4025cf,%esi
> 40101f: b8 00 00 00 00          mov    $0x0,%eax
> 401024: e8 c7 fb ff ff          callq  400bf0 <__isoc99_sscanf@plt>
> 401029: 83 f8 02                cmp    $0x2,%eax
> 40102c: 75 07                   jne    401035 <phase_4+0x29>
> 40102e: 83 7c 24 08 0e          cmpl   $0xe,0x8(%rsp)
> 401033: 76 05                   jbe    40103a <phase_4+0x2e>
> 401035: e8 00 04 00 00          callq  40143a <explode_bomb>
> ```

类似于 *phase_3*，执行完地址为 *0x401024* 的函数后，此时 *%rsp* 值为 *0x7fff-ffff-dee0*，输入的两个值分别存储在栈地址 *0x7fff-ffff-dee8* 和 *0x7fff-ffff-deec* 处（前者存储第一个值）。程序运行至 *0x401029* 处，将 *%eax* 的值与 *2* 相比较，若其不等于 *2*，将跳转至 *0x401035* 执行 *explode_bomb* 函数。所以，**输入应严格限制为两个值**。程序运行至 *0x40102e* 处，比较输入的第一个值与 *0xe* 的大小，当其小于等于 *0xe* 时，程序跳转至 *0x40103a*，否则程序运行 *explode_bomb* 函数。所以，**输入的第一个值应小于等于 *0xe***。

```
40103a: ba 0e 00 00 00          mov    $0xe,%edx
40103f: be 00 00 00 00          mov    $0x0,%esi
401044: 8b 7c 24 08             mov    0x8(%rsp),%edi
401048: e8 81 ff ff ff          callq  400fce <func4>
```

程序跳转至 *0x40103a* 处继续运行，执行调用函数前的参数赋值操作，即分别给 *%edi* ， *%esi* 和 *%edx* 赋值，调用函数 *func4* 。

```
0000000000400fce <func4>:
400fce: 48 83 ec 08             sub    $0x8,%rsp
400fd2: 89 d0                   mov    %edx,%eax
400fd4: 29 f0                   sub    %esi,%eax
400fd6: 89 c1                   mov    %eax,%ecx
400fd8: c1 e9 1f                shr    $0x1f,%ecx
400fdb: 01 c8                   add    %ecx,%eax
400fdd: d1 f8                   sar    %eax
400fdf: 8d 0c 30                lea    (%rax,%rsi,1),%ecx
400fe2: 39 f9                   cmp    %edi,%ecx
400fe4: 7e 0c                   jle    400ff2 <func4+0x24>
400fe6: 8d 51 ff                lea    -0x1(%rcx),%edx
400fe9: e8 e0 ff ff ff          callq  400fce <func4>
400fee: 01 c0                   add    %eax,%eax
400ff0: eb 15                   jmp    401007 <func4+0x39>
400ff2: b8 00 00 00 00          mov    $0x0,%eax
400ff7: 39 f9                   cmp    %edi,%ecx
400ff9: 7d 0c                   jge    401007 <func4+0x39>
400ffb: 8d 71 01                lea    0x1(%rcx),%esi
400ffe: e8 cb ff ff ff          callq  400fce <func4>
401003: 8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
401007: 48 83 c4 08             add    $0x8,%rsp
40100b: c3                      retq
```

在函数 *func4* 起始位置 *0x400fce* 处设置断点，打印出此时部分寄存器值如下：

REG	VAL	备注
%rdi	1	输入参数，非定值，输入字符串的第一个数（以 *x* 表示）
%rsi	0	输入参数，定值
%rdx	0xe	输入参数，定值
%rsp	0x7fff-ffff-ded8	非输入参数

将上述汇编代码可翻译为 *c* 代码如下：

```c
int func4(edi, esi, edx)
{
 eax = (edx - esi) + ((edx - esi) >> 31);
 eax = eax >> 1;
 ecx = rax + rsi;
 if (ecx <= x){
     eax = 0;
     if (ecx >= x)
         return eax;
     else{
         esi = rcx + 1;
```

```
        func4(edi, esi, edx);
        eax = 2 * eax + 1;
        return eax;
    }
 }else{
    edx = rcx - 1;
    func4(edi, esi, edx);
    eax = 2 * eax;
    return eax;
 }
}
```

可以看出，在执行第一个 *if* 条件判断（对应汇编代码地址 *0x400fe4* 处）前，*%ecx* 为定值 *7* ，若 *x* 值为 *7* ，则可返回 *%rax=0* 。

```
401048: e8 81 ff ff ff          callq  400fce <func4>
40104d: 85 c0                   test   %eax,%eax
40104f: 75 07                   jne    401058 <phase_4+0x4c>
401051: 83 7c 24 0c 00          cmpl   $0x0,0xc(%rsp)
401056: 74 05                   je     40105d <phase_4+0x51>
401058: e8 dd 03 00 00          callq  40143a <explode_bomb>
40105d: 48 83 c4 18             add    $0x18,%rsp
401061: c3                      retq
```

回到函数 *phase_4* 调用完函数 *func4* 的位置，即地址 *0x40104d* 处，该语句判断返回值 *%eax* 是否为 *0* ，若不为 *0* ，则跳转到 *0x401058* 处执行 *explode_bomb* 函数。所以，**函数 *func4* 的返回值应为 *0* 。**程序继续执行至 *0x401051* 处，判断输入字符串的第二个值是否为 *0* ，若为 *0* ，则程序顺利运行，否则会执行 *explode_bomb* 函数。所以，**可以正确输入的字符串为 *"7,0"* 。**因为函数 *func4* 会迭代运行，我没有深入考虑其他情况，不排除可能会存在第一个数不为 *7* 而函数 *func4* 同样返回 *0* 的可能性，所以可能存在其他答案。

---

phase_5

```
0000000000401062 <phase_5>:
  401062:   53                      push   %rbx
  401063:   48 83 ec 20             sub    $0x20,%rsp
  401067:   48 89 fb                mov    %rdi,%rbx
  40106a:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  401071:   00 00
  401073:   48 89 44 24 18          mov    %rax,0x18(%rsp)
  401078:   31 c0                   xor    %eax,%eax
  40107a:   e8 9c 02 00 00          callq  40131b <string_length>
  40107f:   83 f8 06                cmp    $0x6,%eax
  401082:   74 4e                   je     4010d2 <phase_5+0x70>
  401084:   e8 b1 03 00 00          callq  40143a <explode_bomb>
  401089:   eb 47                   jmp    4010d2 <phase_5+0x70>
  40108b:   0f b6 0c 03             movzbl (%rbx,%rax,1),%ecx
  40108f:   88 0c 24                mov    %cl,(%rsp)
  401092:   48 8b 14 24             mov    (%rsp),%rdx
  401096:   83 e2 0f                and    $0xf,%edx
  401099:   0f b6 92 b0 24 40 00    movzbl 0x4024b0(%rdx),%edx
  4010a0:   88 54 04 10             mov    %dl,0x10(%rsp,%rax,1)
  4010a4:   48 83 c0 01             add    $0x1,%rax
  4010a8:   48 83 f8 06             cmp    $0x6,%rax
  4010ac:   75 dd                   jne    40108b <phase_5+0x29>
```

```
4010ae:   c6 44 24 16 00          movb   $0x0,0x16(%rsp)
4010b3:   be 5e 24 40 00          mov    $0x40245e,%esi
4010b8:   48 8d 7c 24 10          lea    0x10(%rsp),%rdi
4010bd:   e8 76 02 00 00          callq  401338 <strings_not_equal>
4010c2:   85 c0                   test   %eax,%eax
4010c4:   74 13                   je     4010d9 <phase_5+0x77>
4010c6:   e8 6f 03 00 00          callq  40143a <explode_bomb>
4010cb:   0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
4010d0:   eb 07                   jmp    4010d9 <phase_5+0x77>
4010d2:   b8 00 00 00 00          mov    $0x0,%eax
4010d7:   eb b2                   jmp    40108b <phase_5+0x29>
4010d9:   48 8b 44 24 18          mov    0x18(%rsp),%rax
4010de:   64 48 33 04 25 28 00    xor    %fs:0x28,%rax
4010e5:   00 00
4010e7:   74 05                   je     4010ee <phase_5+0x8c>
4010e9:   e8 42 fa ff ff          callq  400b30 <__stack_chk_fail@plt>
4010ee:   48 83 c4 20             add    $0x20,%rsp
4010f2:   5b                      pop    %rbx
4010f3:   c3                      retq
```

本题目应输入包含六个字符的字符串（下边有具体原因说明），我们输入字符串仍为 *"abcdef"* 进行调试。在程序地址 *0x401062* 处设置断点，此时寄存器 *%rax* 的值为 *0x6038c0* ，指向输入字符串的首地址；*%rsp* 的值为 *0x7fff-ffff-def8* 。

```
(gdb) p (char *) $rax
$1 = 0x6038c0 <input_strings+320> "abcdef"
```

```
401062:   53                      push   %rbx
401063:   48 83 ec 20             sub    $0x20,%rsp
401067:   48 89 fb                mov    %rdi,%rbx
40106a:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
401071:   00 00
401073:   48 89 44 24 18          mov    %rax,0x18(%rsp)
401078:   31 c0                   xor    %eax,%eax
40107a:   e8 9c 02 00 00          callq  40131b <string_length>
40107f:   83 f8 06                cmp    $0x6,%eax
401082:   74 4e                   je     4010d2 <phase_5+0x70>
401084:   e8 b1 03 00 00          callq  40143a <explode_bomb>
```

程序运行至地址 *0x40106a* ，将值 *%fs:0x28* 移至寄存器 *%rax* 中，然后将其值移动到运行栈中提供栈保护机制。查看其值如下：

```
(gdb) p /x $rax
$2 = 0x2a26daffbefa3a00
```

语句 `xor %eax,%eax` 为将 *%eax* 清零的操作。程序地址 *0x40107a* 调用函数 *string_length* ，猜测为求字符串的长度，且函数只有一个输入参数 *%rdi* ，此时 *%rdi* 的值为指向输入字符串的首地址，所以，该处函数的作用应为求输入字符串的长度。程序地址 *0x40107f* 将返回值 *%eax* 与 *6* 相比较，当两者相等时，程序跳转到地址 *0x4010d2* 处，否则将执行 *explode_bomb* 函数。此时栈中值如下：

Val	%rsp
??	0x7fff-ffff-def8
rbx	0x7fff-ffff-def0
0x2a26daffbefa3a00	0x7fff-ffff-dee8
??	0x7fff-ffff-dee0
??	0x7fff-ffff-ded8
??	0x7fff-ffff-ded0

> What mov %fs:0x28,%rax is really doing?
>
> > On x86_64, segmented addressing is no longer used, but the both the FS and GS registers can be used as base-pointer addresses in order to access special operating system data-structures. So what you're seeing is a value loaded at an offset from the value held in the FS register, and not bit manipulation of the contents of the FS register.
> >
> > Specifically what's taking place, is that FS:0x28 on Linux is storing a special sentinel stack-guard value, and the code is performing a stack-guard check. For instance, if you look further in your code, you'll see that the value at FS:0x28 is stored on the stack, and then the contents of the stack are recalled and an XOR is performed with the original value at FS:0x28. If the two values are equal, which means that the zero-bit has been set because XOR'ing two of the same values results in a zero-value, then we jump to the test routine, otherwise we jump to a special function that indicates that the stack was somehow corrupted, and the sentinel value stored on the stack was changed.
>
> 摘抄自：https://stackoverflow.com/questions/10325713/why-does-this-memory-address-fs0x28-fs0x28-have-a-random-value

```
4010d2:   b8 00 00 00 00          mov    $0x0,%eax
4010d7:   eb b2                   jmp    40108b <phase_5+0x29>
```

程序跳转至地址 *0x4010d2* 处，将寄存器 *%eax* 赋值为 *0* ，接着跳转至 *0x4010d7* 。

```
40108b:   0f b6 0c 03             movzbl (%rbx,%rax,1),%ecx
40108f:   88 0c 24                mov    %cl,(%rsp)
401092:   48 8b 14 24             mov    (%rsp),%rdx
401096:   83 e2 0f                and    $0xf,%edx
401099:   0f b6 92 b0 24 40 00    movzbl 0x4024b0(%rdx),%edx
4010a0:   88 54 04 10             mov    %dl,0x10(%rsp,%rax,1)
4010a4:   48 83 c0 01             add    $0x1,%rax
4010a8:   48 83 f8 06             cmp    $0x6,%rax
4010ac:   75 dd                   jne    40108b <phase_5+0x29>
```

程序跳转至地址 *0x40108b* 处，此时寄存器 *%rbx* 保存着输入字符串的首地址。`movzbl (%rbx,%rax,1),%ecx` 此语句是将输入字符串的第 *%rax* 个字符赋值给 *%ecx* ，接下来存储到 *%rsp* 对应的位置，并在此赋值到 *%rdx* 中，以字符串第一个值 *"a"* 为例，此时 *%rdx* 的值为 *0x61*（对应字符的ASCII码）。程序地址 *0x401096* 处，取 *%rdx* 低四位的值存储在 *%edx* 中，此时 *%edx* 值为 *0x01* 。接下来，以地址 *0x4024b0* 为基，*%rdx* 为偏移量的地址处存储的字符（从下边可以看出仍为 *'a'* ）赋值到 *%edx* 中，然后取该字符（*%dl*）赋值到以 *%rsp+0x10* 为基地址，*%rax* 为偏移的运行栈地址中，本例是将 *0x61* 赋值到 *0x7fff-ffff-ded0 + 0x10* 中（此时 *%rax* 为 *0* ）。将 *rax* 的值加 *1* ，并与 *6* 相比较，若不等于 *6* ，则重新跳转至地址 *0x40108b* 处循环运行。所以，该段代码的作用为对输入字符串包含的六个字符，依次取每个字符 *ASCII* 码的低四位为索引，取另外给定的字符串中的相应字符，并将其依次存储在运行时栈中。

地址 *0x4024b0* 处保存的字符串为：

```
(gdb) p (char *) 0x4024b0
$20 = 0x4024b0 <array> "maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?"
```

全部循环遍历后，此时栈中值如下：

Val								%rsp
??								0x7fff-ffff-def8
rbx								0x7fff-ffff-def0
0x2a26daffbefa3a00								0x7fff-ffff-dee8
?	?	r	e	i	u	d	a	0x7fff-ffff-dee0
??								0x7fff-ffff-ded8
?	?	?	?	?	?	?	f - 0x66	0x7fff-ffff-ded0

```
4010ae:   c6 44 24 16 00          movb   $0x0,0x16(%rsp)
4010b3:   be 5e 24 40 00          mov    $0x40245e,%esi
4010b8:   48 8d 7c 24 10          lea    0x10(%rsp),%rdi
4010bd:   e8 76 02 00 00          callq  401338 <strings_not_equal>
4010c2:   85 c0                   test   %eax,%eax
4010c4:   74 13                   je     4010d9 <phase_5+0x77>
4010c6:   e8 6f 03 00 00          callq  40143a <explode_bomb>
```

上一步程序循环赋值结束后，程序执行至地址 *0x4010ae* 处，栈中字节 *0x7fff-ffff-dee6* 处，赋值为 *0*；然后将 *%rsi* 赋值为 *0x40245e*，*%rdi* 赋值为 *0x7fff-ffff-dee0*，指向赋值到栈中字符串的首地址；接下来调用函数 *strings_not_equal* 判断两段字符串是否相等，若相等，则跳转至程序地址 *0x4010d9* 处，否则执行 *explode_bomb* 函数。打印出地址 *0x40245e* 处存放的字符串如下：

```
(gdb) p (char *) 0x40245e
$21 = 0x40245e "flyers"
```

所以，应保证此时存放在栈中的字符串为 ***"flyers"***。其在给定字符串中的索引值依次为：*"9,15,14,5,6,7"*，应取相应字符的ASCII码为：*"0x?9, 0x?f, 0x?e, 0x?5, 0x?6, 0x?7"*，取 *?* 值为 *6* 时对应字符串为：***"ionefg"***。接下来程序可正确运行。

---

phase_6

```
00000000004010f4 <phase_6>:
  4010f4:   41 56                   push   %r14
  4010f6:   41 55                   push   %r13
  4010f8:   41 54                   push   %r12
  4010fa:   55                      push   %rbp
  4010fb:   53                      push   %rbx
  4010fc:   48 83 ec 50             sub    $0x50,%rsp
  401100:   49 89 e5                mov    %rsp,%r13
  401103:   48 89 e6                mov    %rsp,%rsi
  401106:   e8 51 03 00 00          callq  40145c <read_six_numbers>
  40110b:   49 89 e6                mov    %rsp,%r14
  40110e:   41 bc 00 00 00 00       mov    $0x0,%r12d
  401114:   4c 89 ed                mov    %r13,%rbp
  401117:   41 8b 45 00             mov    0x0(%r13),%eax
  40111b:   83 e8 01                sub    $0x1,%eax
  40111e:   83 f8 05                cmp    $0x5,%eax
  401121:   76 05                   jbe    401128 <phase_6+0x34>
  401123:   e8 12 03 00 00          callq  40143a <explode_bomb>
  401128:   41 83 c4 01             add    $0x1,%r12d
```

```
40112c:   41 83 fc 06             cmp    $0x6,%r12d
401130:   74 21                   je     401153 <phase_6+0x5f>
401132:   44 89 e3                mov    %r12d,%ebx
401135:   48 63 c3                movslq %ebx,%rax
401138:   8b 04 84                mov    (%rsp,%rax,4),%eax
40113b:   39 45 00                cmp    %eax,0x0(%rbp)
40113e:   75 05                   jne    401145 <phase_6+0x51>
401140:   e8 f5 02 00 00          callq  40143a <explode_bomb>
401145:   83 c3 01                add    $0x1,%ebx
401148:   83 fb 05                cmp    $0x5,%ebx
40114b:   7e e8                   jle    401135 <phase_6+0x41>
40114d:   49 83 c5 04             add    $0x4,%r13
401151:   eb c1                   jmp    401114 <phase_6+0x20>
401153:   48 8d 74 24 18          lea    0x18(%rsp),%rsi
401158:   4c 89 f0                mov    %r14,%rax
40115b:   b9 07 00 00 00          mov    $0x7,%ecx
401160:   89 ca                   mov    %ecx,%edx
401162:   2b 10                   sub    (%rax),%edx
401164:   89 10                   mov    %edx,(%rax)
401166:   48 83 c0 04             add    $0x4,%rax
40116a:   48 39 f0                cmp    %rsi,%rax
40116d:   75 f1                   jne    401160 <phase_6+0x6c>
40116f:   be 00 00 00 00          mov    $0x0,%esi
401174:   eb 21                   jmp    401197 <phase_6+0xa3>
401176:   48 8b 52 08             mov    0x8(%rdx),%rdx
40117a:   83 c0 01                add    $0x1,%eax
40117d:   39 c8                   cmp    %ecx,%eax
40117f:   75 f5                   jne    401176 <phase_6+0x82>
401181:   eb 05                   jmp    401188 <phase_6+0x94>
401183:   ba d0 32 60 00          mov    $0x6032d0,%edx
401188:   48 89 54 74 20          mov    %rdx,0x20(%rsp,%rsi,2)
40118d:   48 83 c6 04             add    $0x4,%rsi
401191:   48 83 fe 18             cmp    $0x18,%rsi
401195:   74 14                   je     4011ab <phase_6+0xb7>
401197:   8b 0c 34                mov    (%rsp,%rsi,1),%ecx
40119a:   83 f9 01                cmp    $0x1,%ecx
40119d:   7e e4                   jle    401183 <phase_6+0x8f>
40119f:   b8 01 00 00 00          mov    $0x1,%eax
4011a4:   ba d0 32 60 00          mov    $0x6032d0,%edx
4011a9:   eb cb                   jmp    401176 <phase_6+0x82>
4011ab:   48 8b 5c 24 20          mov    0x20(%rsp),%rbx
4011b0:   48 8d 44 24 28          lea    0x28(%rsp),%rax
4011b5:   48 8d 74 24 50          lea    0x50(%rsp),%rsi
4011ba:   48 89 d9                mov    %rbx,%rcx
4011bd:   48 8b 10                mov    (%rax),%rdx
4011c0:   48 89 51 08             mov    %rdx,0x8(%rcx)
4011c4:   48 83 c0 08             add    $0x8,%rax
4011c8:   48 39 f0                cmp    %rsi,%rax
4011cb:   74 05                   je     4011d2 <phase_6+0xde>
4011cd:   48 89 d1                mov    %rdx,%rcx
4011d0:   eb eb                   jmp    4011bd <phase_6+0xc9>
4011d2:   48 c7 42 08 00 00 00    movq   $0x0,0x8(%rdx)
4011d9:   00
4011da:   bd 05 00 00 00          mov    $0x5,%ebp
4011df:   48 8b 43 08             mov    0x8(%rbx),%rax
4011e3:   8b 00                   mov    (%rax),%eax
4011e5:   39 03                   cmp    %eax,(%rbx)
4011e7:   7d 05                   jge    4011ee <phase_6+0xfa>
```

```
4011e9:   e8 4c 02 00 00          callq  40143a <explode_bomb>
4011ee:   48 8b 5b 08             mov    0x8(%rbx),%rbx
4011f2:   83 ed 01                sub    $0x1,%ebp
4011f5:   75 e8                   jne    4011df <phase_6+0xeb>
4011f7:   48 83 c4 50             add    $0x50,%rsp
4011fb:   5b                      pop    %rbx
4011fc:   5d                      pop    %rbp
4011fd:   41 5c                   pop    %r12
4011ff:   41 5d                   pop    %r13
401201:   41 5e                   pop    %r14
401203:   c3                      retq
```

该题目可以说是这几道题目中难度最大的，重要的是理清复杂的汇编操作背后的简单逻辑关系。与第二题类似，输入参数为包含六个数字的字符串，本次输入 *"1,2,3,4,5,6"* 进行调试。

```
4010f4:   41 56                   push   %r14
4010f6:   41 55                   push   %r13
4010f8:   41 54                   push   %r12
4010fa:   55                      push   %rbp
4010fb:   53                      push   %rbx
4010fc:   48 83 ec 50             sub    $0x50,%rsp
401100:   49 89 e5                mov    %rsp,%r13
401103:   48 89 e6                mov    %rsp,%rsi
401106:   e8 51 03 00 00          callq  40145c <read_six_numbers>
```

程序执行到 *0x401106* 时，寄存器 *%rsp* ， *%13* 和 *%rsi* 的值均为 *0x7fff-ffff-de80* 。执行函数 *read_six_numbers* ，将输入的六个数字赋值到运行时栈的相应位置如下：

Val		%rsp
??		0x7fff-ffff-de98
6	5	0x7fff-ffff-de90
4	3	0x7fff-ffff-de88
2	1	0x7fff-ffff-de80

```
40110b:   49 89 e6                mov    %rsp,%r14
40110e:   41 bc 00 00 00 00       mov    $0x0,%r12d
401114:   4c 89 ed                mov    %r13,%rbp
401117:   41 8b 45 00             mov    0x0(%r13),%eax
40111b:   83 e8 01                sub    $0x1,%eax
40111e:   83 f8 05                cmp    $0x5,%eax
401121:   76 05                   jbe    401128 <phase_6+0x34>
401123:   e8 12 03 00 00          callq  40143a <explode_bomb>
401128:   41 83 c4 01             add    $0x1,%r12d
40112c:   41 83 fc 06             cmp    $0x6,%r12d
401130:   74 21                   je     401153 <phase_6+0x5f>
```

接下来程序将寄存器 *r14* 和 *%rbp* 赋值为栈底地址值，即 *0x7fff-ffffde80* ，将 *r12d* 赋值为 *0* （*%12* 将充当计数器控制循环，下边代码可以看出），将 *%eax* 赋值为输入的第一个数值。程序运行至地址 *0x40111b* 处，*%eax* 的减 *1* ，并与数值 *5* 相比较，当其小于等于 *5* 时，跳转时地址 *0x401128* ，否则执行 *explode_bomb* 函数。所以，**输入的第一个值 *x1* 应满足条件 *x1<=6***。程序跳转至地址 *0x401128* 处，对 *r12d* 值加 *1* ，当其值等于 *6* 时，跳转到地址 *0x401153* ，否则继续向下执行，这里可以猜测 *%r12d* 控制着对这六个数字的循环访问。

```
401132:   44 89 e3              mov    %r12d,%ebx
401135:   48 63 c3              movslq %ebx,%rax
401138:   8b 04 84              mov    (%rsp,%rax,4),%eax
40113b:   39 45 00              cmp    %eax,0x0(%rbp)
40113e:   75 05                 jne    401145 <phase_6+0x51>
401140:   e8 f5 02 00 00        callq  40143a <explode_bomb>
401145:   83 c3 01              add    $0x1,%ebx
401148:   83 fb 05              cmp    $0x5,%ebx
40114b:   7e e8                 jle    401135 <phase_6+0x41>
```

程序运行至地址 *0x401132* 处，寄存器 *%r12d* 值为 *1* ，赋值给 *%ebx* ，再将其赋值给 *%rax* 充当在运行栈中以 *%rsp* 为基地址的偏移量，取出相应的数字赋值给 *%eax* ，该次为取出第二个值与第一个值相比较，若两值相等，则执行函数 *explode_bomb* ；若两者不相等，跳转至地址 *0x401145* 处将 *%ebx* 的值加 *1* ，并判断其值是否小于等于 *5* ，满足条件时继续跳转至地址 *0x401135* 处取下一个数与第一个值相比较。所以，**输入的六个数字中第一个应与其它值相异。**（后边我们会看到以寄存器 *%r12d* 控制的循环至其他数字时，均要求与其后输入的数字相异。所以，输入的六个数字应各不相同）。

```
40114d:   49 83 c5 04           add    $0x4,%r13
401151:   eb c1                 jmp    401114 <phase_6+0x20>
```

将第一个数字与其它数字比较完后，将 *r13* 的值加 *4* ，使其指向输入的第二个数字（此处对应着 *%r12d* 控制的循环），程序跳转至 *0x401114* 处，对第二个数字重复先前的运算。此时我们可以得出结论：**对于输入的每个数字 *x* ，应满足 *x<=6* ，且输入的六个数字各不相同 。**

```
401153:   48 8d 74 24 18        lea    0x18(%rsp),%rsi
401158:   4c 89 f0              mov    %r14,%rax
40115b:   b9 07 00 00 00        mov    $0x7,%ecx
401160:   89 ca                 mov    %ecx,%edx
401162:   2b 10                 sub    (%rax),%edx
401164:   89 10                 mov    %edx,(%rax)
401166:   48 83 c0 04           add    $0x4,%rax
40116a:   48 39 f0              cmp    %rsi,%rax
40116d:   75 f1                 jne    401160 <phase_6+0x6c>
```

对六个数字循环结束后，程序跳转至 *0x401153* 。将第六个数字在栈上下一位的地址（*0x7fff-ffff-de98*）赋值给 *%rsi* ，将第一个数字在栈上地址赋值给 *%rax* ，将值 *7* 赋值给 *%ecx* ，在赋值给 *%edx* ，接着另 *%edx* 减去 *%rax* 在运行时栈上指定的值，并将其结果存入栈上该位置。程序 *0x401166* 处对 *%rax* 值加 *4* ，并判断是否到达 *%rsi* 指定的位置，若两者指向位置不同，则程序跳转至 *0x401160* 循环运行。所以，该段代码的作用为对于运行时栈上保存的输入值 *x* ，分别以 *7-x* 替代之。循环结束后，运行时栈上的值如下：

Val		%rsp
??		0x7fff-ffff-de98
1	2	0x7fff-ffff-de90
3	4	0x7fff-ffff-de88
5	6	0x7fff-ffff-de80

```
40116f:   be 00 00 00 00        mov    $0x0,%esi
401174:   eb 21                 jmp    401197 <phase_6+0xa3>
401176:   48 8b 52 08           mov    0x8(%rdx),%rdx
40117a:   83 c0 01              add    $0x1,%eax
40117d:   39 c8                 cmp    %ecx,%eax
40117f:   75 f5                 jne    401176 <phase_6+0x82>
401181:   eb 05                 jmp    401188 <phase_6+0x94>
401183:   ba d0 32 60 00        mov    $0x6032d0,%edx
```

```
401188:   48 89 54 74 20          mov    %rdx,0x20(%rsp,%rsi,2)
40118d:   48 83 c6 04             add    $0x4,%rsi
401191:   48 83 fe 18             cmp    $0x18,%rsi
401195:   74 14                   je     4011ab <phase_6+0xb7>
401197:   8b 0c 34                mov    (%rsp,%rsi,1),%ecx
40119a:   83 f9 01                cmp    $0x1,%ecx
40119d:   7e e4                   jle    401183 <phase_6+0x8f>
40119f:   b8 01 00 00 00          mov    $0x1,%eax
4011a4:   ba d0 32 60 00          mov    $0x6032d0,%edx
4011a9:   eb cb                   jmp    401176 <phase_6+0x82>
```

程序执行至 *0x40116f* ，将 *%esi* 赋值为 *0* ，并跳转到地址 *0x401197* ，取以 *%rsp* 为基地址，*%rsi* 值为偏移地址

的运行时栈上的值赋值给 *%ecx* ，并与值 *1* 相比较，若其小于等于 *1* ，则跳转至 *0x401183* 处执行，否则程序向下继续执行。此时 *%esi* 为 *0* ，相当于赋值给 *%ecx* 六个值中的第一个，即为 *6* ，其大于 *1* ，继续向下执行。程序地址 *0x40119f* 处，将 *%eax* 赋值为 *1* ，*%edx* 赋值为 *0x6032d0* ，接着程序跳转至 *0x401176* 。

在程序地址 *0x401176* 处，语句 `0x8(%rdx),%rdx` 将 *%rdx* 值加 *8* 的地址处存储的值赋值给 *%rdx* ，接着将 *%eax* 的值加 *1* 后，比较其与 *%ecx* (此时 *%ecx* 为六个值中的第一个，即为 *6*)，如果两者不相等，则重新跳转到 *0x401176* 循环执行该段前述语句。当两者相等时，程序跳转到 *0x401188* 将 *%rdx* 的值赋值到运行时栈中以 *%rsp+0x20* 为基地址，*%rsi∗2* 为偏移地址的位置。接着将 *%rsi* 值加 *4* ，并与 *0x18* 比较，若两者相等，则程序跳转至地址 *0x4011ab* 处，否则程序继续向下执行，回到 *0x401197* 的位置。

此时可以判断该段程序是依次取出运行时栈中的六个数值（每一个表示为 *7-x* ），并取出以 *0x6032d0* 为基地址，以 *7-x* 为偏移量的一部分处所存储的值，将其赋值到运行时栈中相应的位置。相应的对应关系如下表：

Address	Value	7-x
	0x6032d0	0 , 1
0x6032d8	0x6032e0	2
0x6032e8	0x6032f0	3
0x6032f8	0x603300	4
0x603308	0x603310	5
0x603318	0x603320	6
0x603328	0	

该段程序循环运行结束后运行时栈中的值如下：

Val		%rsp
??		0x7fff-ffff-ded0
0x6032d0		0x7fff-ffff-dec8
0x6032e0		0x7fff-ffff-dec0
0x6032f0		0x7fff-ffff-deb8
0x603300		0x7fff-ffff-deb0
0x603310		0x7fff-ffff-dea8
0x603320		0x7fff-ffff-dea0
??		0x7fff-ffff-de98
1	2	0x7fff-ffff-de90
3	4	0x7fff-ffff-de88
5	6	0x7fff-ffff-de80

```
4011ab:   48 8b 5c 24 20          mov    0x20(%rsp),%rbx
4011b0:   48 8d 44 24 28          lea    0x28(%rsp),%rax
4011b5:   48 8d 74 24 50          lea    0x50(%rsp),%rsi
4011ba:   48 89 d9                mov    %rbx,%rcx
4011bd:   48 8b 10                mov    (%rax),%rdx
4011c0:   48 89 51 08             mov    %rdx,0x8(%rcx)
4011c4:   48 83 c0 08             add    $0x8,%rax
4011c8:   48 39 f0                cmp    %rsi,%rax
4011cb:   74 05                   je     4011d2 <phase_6+0xde>
4011cd:   48 89 d1                mov    %rdx,%rcx
4011d0:   eb eb                   jmp    4011bd <phase_6+0xc9>
4011d2:   48 c7 42 08 00 00 00    movq   $0x0,0x8(%rdx)
4011d9:   00
```

程序运行至地址 *0x4011ab* 处，将 *%rbx* 赋值为内存中取到栈中的六个数值的第一个，即 *0x603320*，将 *%rax* 赋值为指向第二个数值的栈地址，即 *0x7fff-ffff-dea8*，将 *%rsi* 赋值为指向最后一个数值下一位的栈地址，即 *0x7fff-ffff-ded0*。程序 *0x4011ba -- 0x4011c0* 三条语句执行的操作为：将运行时栈中栈地址为 *%rax* 中的数值，保存在以 *%rcx* 值加 *8* 为地址的内存中，此时意即将第二个数值保存在以第一个数值加 *8* 为地址的内存中。接着移动指针 *%rax* 的值指向第三个数值的栈地址，对第二个和第三个数值执行类似的操作，即将第三个数值保存在以第二个数值加 *8* 为地址的内存中. . . 直至 *%rax* 值等于 *%rsi*，循环结束，跳转至 *0x4011d2*，为第六个数值加 *8* 为地址的内存赋值为 *0*。运行结束后示意图如下：

Value		Address	Val		%rsp
			??		0x7fff-ffff-ded0
0	<<	0x6032d8	0x6032d0		0x7fff-ffff-dec8
0x6032d0	<<	0x6032e8	0x6032e0		0x7fff-ffff-dec0
0x6032e0	<<	0x6032f8	0x6032f0		0x7fff-ffff-deb8
0x6032f0	<<	0x603308	0x603300		0x7fff-ffff-deb0
0x603300	<<	0x603318	0x603310		0x7fff-ffff-dea8
0x603310	<<	0x603328	0x603320		0x7fff-ffff-dea0
			??		0x7fff-ffff-de98
			1	2	0x7fff-ffff-de90
			3	4	0x7fff-ffff-de88
			5	6	0x7fff-ffff-de80

```
4011da:   bd 05 00 00 00          mov    $0x5,%ebp
4011df:   48 8b 43 08             mov    0x8(%rbx),%rax
4011e3:   8b 00                   mov    (%rax),%eax
4011e5:   39 03                   cmp    %eax,(%rbx)
4011e7:   7d 05                   jge    4011ee <phase_6+0xfa>
4011e9:   e8 4c 02 00 00          callq  40143a <explode_bomb>
4011ee:   48 8b 5b 08             mov    0x8(%rbx),%rbx
4011f2:   83 ed 01                sub    $0x1,%ebp
4011f5:   75 e8                   jne    4011df <phase_6+0xeb>
```

程序执行至 *0x4011da*，将寄存器 *%ebp* 赋值为 *5*（用作计数器），此时 *%rbx* 存储的值为运行时栈中的第一个元素，即 *0x603320*，将其值加 *8* 为内存地址处的值（实为运行时栈中第二个元素的值）取出赋给 *%rax*，接下来取以 *%rax* 为内存地址处的值赋给 *%eax*，并与以 *%rax* 为内存地址处的值作比较，若后者小于前者，则执行 *explode_bomb* 函数，否则跳过该函数将 *%rbx* 的值更新为其值加 *8* 为内存地址处的值（实为运行时栈中第二个元素的值），并将 *%ebp* 值减 *1*，若 *%ebp* 值不为 *0*，程序跳转至 *0x4011df* 循环运行。所以，该段程序的作用时要求**运行时栈中从下往上对应相应内存地址中的值应降序排列**，如此程序可以顺利执行。打印出以运行时栈中元素为内存地址时对应存储的数值如下：

序号	Address	Value	
a	0x6032d0	0x14c	1
b	0x6032e0	0xa8	2
c	0x6032f0	0x39c	3
d	0x603300	0x2b3	4
e	0x603310	0x1dd	5
f	0x603320	0x1bb	6

可以看出各地址对应的值按序号从大到小排序为：*c* > *d* > *e* > *f* > *a* > *b* 。所以，在运行时栈中存储的值应为下表所示，对应所应输入的字符串为：***“4, 3, 2, 1, 6, 5”*** 。

		Val		%rsp
		??		0x7fff-ffff-ded0
0xa8	<<	0x6032e0		0x7fff-ffff-dec8
0x14c	<<	0x6032d0		0x7fff-ffff-dec0
0x1bb	<<	0x603320		0x7fff-ffff-deb8
0x1dd	<<	0x603310		0x7fff-ffff-deb0
0x2b3	<<	0x603300		0x7fff-ffff-dea8
0x39c	<<	0x6032f0		0x7fff-ffff-dea0
		??		0x7fff-ffff-de98
5	6	2	1	0x7fff-ffff-de90
1	2 ←	6	5	0x7fff-ffff-de88
3	4	4	3	0x7fff-ffff-de80