# CA Bomb Lab Assignment

- We start off by setting a breakpoint to the <explode_bomb> function to avoid explosions, using the break command.

## Phase 1:

```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x00000000400e8d <+0>:     sub    $0x8,%rsp
   0x00000000400e91 <+4>:     mov    $0x4023d0,%esi
   0x00000000400e96 <+9>:     call   0x401345 <strings_not_equal>
   0x00000000400e9b <+14>:    test   %eax,%eax
   0x00000000400e9d <+16>:    je     0x400ea4 <phase_1+23>
   0x00000000400e9f <+18>:    call   0x401444 <explode_bomb>
   0x00000000400ea4 <+23>:    add    $0x8,%rsp
   0x00000000400ea8 <+27>:    ret
End of assembler dump.
(gdb) disas strings_not_equal
Dump of assembler code for function strings_not_equal:
   0x0000000000401345 <+0>:     push   %r12
   0x0000000000401347 <+2>:     push   %rbp
   0x0000000000401348 <+3>:     push   %rbx
   0x0000000000401349 <+4>:     mov    %rdi,%rbx
   0x000000000040134c <+7>:     mov    %rsi,%rbp
   0x000000000040134f <+10>:    call   0x401327 <string_length>
   0x0000000000401354 <+15>:    mov    %eax,%r12d
   0x0000000000401357 <+18>:    mov    %rbp,%rdi
   0x000000000040135a <+21>:    call   0x401327 <string_length>
   0x000000000040135f <+26>:    mov    $0x1,%edx
   0x0000000000401364 <+31>:    cmp    %eax,%r12d
   0x0000000000401367 <+34>:    jne    0x4013a5 <strings_not_equal+96>
   0x0000000000401369 <+36>:    movzbl (%rbx),%eax
   0x000000000040136c <+39>:    test   %al,%al
   0x000000000040136e <+41>:    je     0x401392 <strings_not_equal+77>
   0x0000000000401370 <+43>:    cmp    0x0(%rbp),%al
   0x0000000000401373 <+46>:    je     0x40137c <strings_not_equal+55>
   0x0000000000401375 <+48>:    jmp    0x401399 <strings_not_equal+84>
   0x0000000000401377 <+50>:    cmp    0x0(%rbp),%al
   0x000000000040137a <+53>:    jne    0x4013a0 <strings_not_equal+91>
   0x000000000040137c <+55>:    add    $0x1,%rbx
   0x0000000000401380 <+59>:    add    $0x1,%rbp
   0x0000000000401384 <+63>:    movzbl (%rbx),%eax
   0x0000000000401387 <+66>:    test   %al,%al
   0x0000000000401389 <+68>:    jne    0x401377 <strings_not_equal+50>
   0x000000000040138b <+70>:    mov    $0x0,%edx
   0x0000000000401390 <+75>:    jmp    0x4013a5 <strings_not_equal+96>
   0x0000000000401392 <+77>:    mov    $0x0,%edx
   0x0000000000401397 <+82>:    jmp    0x4013a5 <strings_not_equal+96>
   0x0000000000401399 <+84>:    mov    $0x1,%edx
   0x000000000040139e <+89>:    jmp    0x4013a5 <strings_not_equal+96>
   0x00000000004013a0 <+91>:    mov    $0x1,%edx
   0x00000000004013a5 <+96>:    mov    %edx,%eax
   0x00000000004013a7 <+98>:    pop    %rbx
   0x00000000004013a8 <+99>:    pop    %rbp
   0x00000000004013a9 <+100>:   pop    %r12
   0x00000000004013ab <+102>:   ret
End of assembler dump.
```

- By looking at the assambly code of <phase_1> we can say that the explosion will depend on the return value of <strings_not_equal> function.
- In the <strings_not_equal> function we see that the input string is being compared to the string in the register rbp.
- To access this we set a break point at <string_length> and run the program, when the program stops at the function <string_length> we use ` x/s $rbp ` command to get the string at rbp.

```
(gdb) x/s $rbp
0x4023d0:       "I was trying to give Tina Fey more material."
```

So the first phase can be diffused by using the string:
"I was trying to give Tina Fey more material."

## Phase 2:

```
(gdb) disas phase_2
Dump of assembler code for function phase_2:
   0x0000000000400ea9 <+0>:     push   %rbp
   0x0000000000400eaa <+1>:     push   %rbx
   0x0000000000400eab <+2>:     sub    $0x28,%rsp
   0x0000000000400eaf <+6>:     mov    %fs:0x28,%rax
   0x0000000000400eb8 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400ebd <+20>:    xor    %eax,%eax
   0x0000000000400ebf <+22>:    mov    %rsp,%rsi
   0x0000000000400ec2 <+25>:    call   0x401466 <read_six_numbers>
   0x0000000000400ec7 <+30>:    cmpl   $0x0,(%rsp)
   0x0000000000400ecb <+34>:    jns    0x400ed2 <phase_2+41>
   0x0000000000400ecd <+36>:    call   0x401444 <explode_bomb>
   0x0000000000400ed2 <+41>:    mov    %rsp,%rbp
   0x0000000000400ed5 <+44>:    mov    $0x1,%ebx
   0x0000000000400eda <+49>:    mov    %ebx,%eax
   0x0000000000400edc <+51>:    add    0x0(%rbp),%eax
   0x0000000000400edf <+54>:    cmp    %eax,0x4(%rbp)
   0x0000000000400ee2 <+57>:    je     0x400ee9 <phase_2+64>
   0x0000000000400ee4 <+59>:    call   0x401444 <explode_bomb>
   0x0000000000400ee9 <+64>:    add    $0x1,%ebx
   0x0000000000400eec <+67>:    add    $0x4,%rbp
   0x0000000000400ef0 <+71>:    cmp    $0x6,%ebx
   0x0000000000400ef3 <+74>:    jne    0x400eda <phase_2+49>
   0x0000000000400ef5 <+76>:    mov    0x18(%rsp),%rax
   0x0000000000400efa <+81>:    xor    %fs:0x28,%rax
   0x0000000000400f03 <+90>:    je     0x400f0a <phase_2+97>
   0x0000000000400f05 <+92>:    call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000400f0a <+97>:    add    $0x28,%rsp
   0x0000000000400f0e <+101>:   pop    %rbx
   0x0000000000400f0f <+102>:   pop    %rbp
   0x0000000000400f10 <+103>:   ret
End of assembler dump.
```

- From the above code we can say that there is a loop running and the register (ebx) is being used to keep track of number of loops ran so far.
- We can also say that the value in eax is used for comparison with the inputs given which were stored in (rbp).
- In each loop the value of (eax) is incremented by the value of (ebx), since the initial value was 1 we can say that the values to be given as inputs are: 1, 2, 4, 7, 11, 16.

So the secound phase can be diffused by using the integers: “1 2 4 7 11 16”

## Phase 3:

```
(gdb) disas phase_3
Dump of assembler code for function phase_3:
   0x0000000000400f11 <+0>:     sub    $0x18,%rsp
   0x0000000000400f15 <+4>:     mov    %fs:0x28,%rax
   0x0000000000400f1e <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000400f23 <+18>:    xor    %eax,%eax
   0x0000000000400f25 <+20>:    lea    0x4(%rsp),%rcx
   0x0000000000400f2a <+25>:    mov    %rsp,%rdx
   0x0000000000400f2d <+28>:    mov    $0x4025cf,%esi
   0x0000000000400f32 <+33>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x0000000000400f37 <+38>:    cmp    $0x1,%eax
   0x0000000000400f3a <+41>:    jg     0x400f41 <phase_3+48>
   0x0000000000400f3c <+43>:    call   0x401444 <explode_bomb>
   0x0000000000400f41 <+48>:    cmpl   $0x7,(%rsp)
   0x0000000000400f45 <+52>:    ja     0x400fac <phase_3+155>
   0x0000000000400f47 <+54>:    mov    (%rsp),%eax
   0x0000000000400f4a <+57>:    jmp    *0x402440(,%rax,8)
   0x0000000000400f51 <+64>:    mov    $0x2af,%eax
   0x0000000000400f56 <+69>:    jmp    0x400f5d <phase_3+76>
   0x0000000000400f58 <+71>:    mov    $0x0,%eax
   0x0000000000400f5d <+76>:    sub    $0x165,%eax
   0x0000000000400f62 <+81>:    jmp    0x400f69 <phase_3+88>
   0x0000000000400f64 <+83>:    mov    $0x0,%eax
   0x0000000000400f69 <+88>:    add    $0x16d,%eax
   0x0000000000400f6e <+93>:    jmp    0x400f75 <phase_3+100>
   0x0000000000400f70 <+95>:    mov    $0x0,%eax
   0x0000000000400f75 <+100>:   sub    $0x18c,%eax
   0x0000000000400f7a <+105>:   jmp    0x400f81 <phase_3+112>
   0x0000000000400f7c <+107>:   mov    $0x0,%eax
   0x0000000000400f81 <+112>:   add    $0x18c,%eax
   0x0000000000400f86 <+117>:   jmp    0x400f8d <phase_3+124>
   0x0000000000400f88 <+119>:   mov    $0x0,%eax
   0x0000000000400f8d <+124>:   sub    $0x18c,%eax
   0x0000000000400f92 <+129>:   jmp    0x400f99 <phase_3+136>
   0x0000000000400f94 <+131>:   mov    $0x0,%eax
   0x0000000000400f99 <+136>:   add    $0x18c,%eax
   0x0000000000400f9e <+141>:   jmp    0x400fa5 <phase_3+148>
   0x0000000000400fa0 <+143>:   mov    $0x0,%eax
   0x0000000000400fa5 <+148>:   sub    $0x18c,%eax
   0x0000000000400faa <+153>:   jmp    0x400fb6 <phase_3+165>
   0x0000000000400fac <+155>:   call   0x401444 <explode_bomb>
   0x0000000000400fb1 <+160>:   mov    $0x0,%eax
   0x0000000000400fb6 <+165>:   cmpl   $0x5,(%rsp)
   0x0000000000400fba <+169>:   jg     0x400fc2 <phase_3+177>
   0x0000000000400fbc <+171>:   cmp    0x4(%rsp),%eax
   0x0000000000400fc0 <+175>:   je     0x400fc7 <phase_3+182>
   0x0000000000400fc2 <+177>:   call   0x401444 <explode_bomb>
   0x0000000000400fc7 <+182>:   mov    0x8(%rsp),%rax
   0x0000000000400fcc <+187>:   xor    %fs:0x28,%rax
   0x0000000000400fd5 <+196>:   je     0x400fdc <phase_3+203>
   0x0000000000400fd7 <+198>:   call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000400fdc <+203>:   add    $0x18,%rsp
   0x0000000000400fe0 <+207>:   ret
End of assembler dump.
```

- From the above code we can say that it has a switch case.
- By using the ‘x/s 0x4025cf’ command we get to know that we have to give two numbers as input which will be stored at (rsp) and (rsp+4).

- By analysing the code we get to know that the first input is case value, depending on the which the value of (eax) changes, and the secound input is being compared to value in (eax).
- To solve this we create a break point at the address ‘0x0000000000400fbc’ (the address of the comparison) and run it giving the first input as some number below 7 (Here we will use 2).
- when the program breaks at ‘0x0000000000400fbc’ we use the ‘print (int ) ($eax)’ command to check the value at (eax) (in our case it is -31).

There are multiple answers to phase 3 one of the ways to diffuse it is giving it the input “2 -31”.

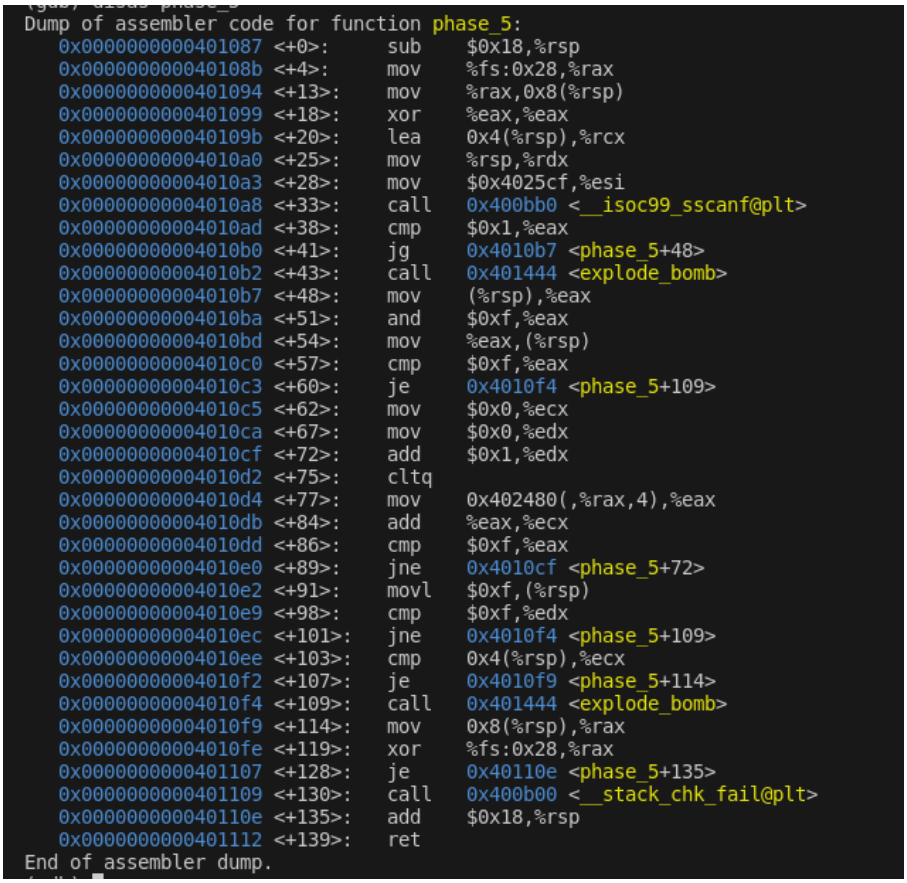
## Phase 4:

```
Dump of assembler code for function phase_4:
   0x0000000000401014 <+0>:     sub    $0x18,%rsp
   0x0000000000401018 <+4>:     mov    %fs:0x28,%rax
   0x0000000000401021 <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000401026 <+18>:    xor    %eax,%eax
   0x0000000000401028 <+20>:    lea    0x4(%rsp),%rcx
   0x000000000040102d <+25>:    mov    %rsp,%rdx
   0x0000000000401030 <+28>:    mov    $0x4025cf,%esi
   0x0000000000401035 <+33>:    call   0x400bb0 <__isoc99_sscanf@plt>
   0x000000000040103a <+38>:    cmp    $0x2,%eax
   0x000000000040103d <+41>:    jne    0x401045 <phase_4+49>
   0x000000000040103f <+43>:    cmpl   $0xe,(%rsp)
   0x0000000000401043 <+47>:    jbe    0x40104a <phase_4+54>
   0x0000000000401045 <+49>:    call   0x401444 <explode_bomb>
   0x000000000040104a <+54>:    mov    $0xe,%edx
   0x000000000040104f <+59>:    mov    $0x0,%esi
   0x0000000000401054 <+64>:    mov    (%rsp),%edi
   0x0000000000401057 <+67>:    call   0x400fe1 <func4>
   0x000000000040105c <+72>:    cmp    $0x1f,%eax
   0x000000000040105f <+75>:    jne    0x401068 <phase_4+84>
   0x0000000000401061 <+77>:    cmpl   $0x1f,0x4(%rsp)
   0x0000000000401066 <+82>:    je     0x40106d <phase_4+89>
   0x0000000000401068 <+84>:    call   0x401444 <explode_bomb>
   0x000000000040106d <+89>:    mov    0x8(%rsp),%rax
   0x0000000000401072 <+94>:    xor    %fs:0x28,%rax
   0x000000000040107b <+103>:   je     0x401082 <phase_4+110>
   0x000000000040107d <+105>:   call   0x400b00 <__stack_chk_fail@plt>
   0x0000000000401082 <+110>:   add    $0x18,%rsp
   0x0000000000401086 <+114>:   ret
End of assembler dump.
(gdb) disas func4
Dump of assembler code for function func4:
   0x0000000000400fe1 <+0>:     push   %rbx
   0x0000000000400fe2 <+1>:     mov    %edx,%eax
   0x0000000000400fe4 <+3>:     sub    %esi,%eax
   0x0000000000400fe6 <+5>:     mov    %eax,%ebx
   0x0000000000400fe8 <+7>:     shr    $0x1f,%ebx
   0x0000000000400feb <+10>:    add    %ebx,%eax
   0x0000000000400fed <+12>:    sar    %eax
   0x0000000000400fef <+14>:    lea    (%rax,%rsi,1),%ebx
   0x0000000000400ff2 <+17>:    cmp    %edi,%ebx
   0x0000000000400ff4 <+19>:    jle    0x401002 <func4+33>
   0x0000000000400ff6 <+21>:    lea    -0x1(%rbx),%edx
   0x0000000000400ff9 <+24>:    call   0x400fe1 <func4>
   0x0000000000400ffe <+29>:    add    %ebx,%eax
   0x0000000000401000 <+31>:    jmp    0x401012 <func4+49>
   0x0000000000401002 <+33>:    mov    %ebx,%eax
   0x0000000000401004 <+35>:    cmp    %edi,%ebx
   0x0000000000401006 <+37>:    jge    0x401012 <func4+49>
   0x0000000000401008 <+39>:    lea    0x1(%rbx),%esi
   0x000000000040100b <+42>:    call   0x400fe1 <func4>
   0x0000000000401010 <+47>:    add    %ebx,%eax
   0x0000000000401012 <+49>:    pop    %rbx
   0x0000000000401013 <+50>:    ret
End of assembler dump.
```

- From the above code we can say that it has a recursive function.
- By using the ‘x/s 0x4025cf’ command we get to know that we have to give two numbers as input which will be stored at (rsp) and (rsp+4).
- In the code we can see ‘cmpl $0x1f,0x4(%rsp)’ command from which we can say that the secound input is 31(decimal representation of 0x1f), we can also say that the value of first input is less than 14.
- From the <func4> code we can say that 7 is the minimum value of the first input. So by trial and error we can say find that the value of first input is 13.

So the fourth phase can be diffused by using the integers:
“13 31”

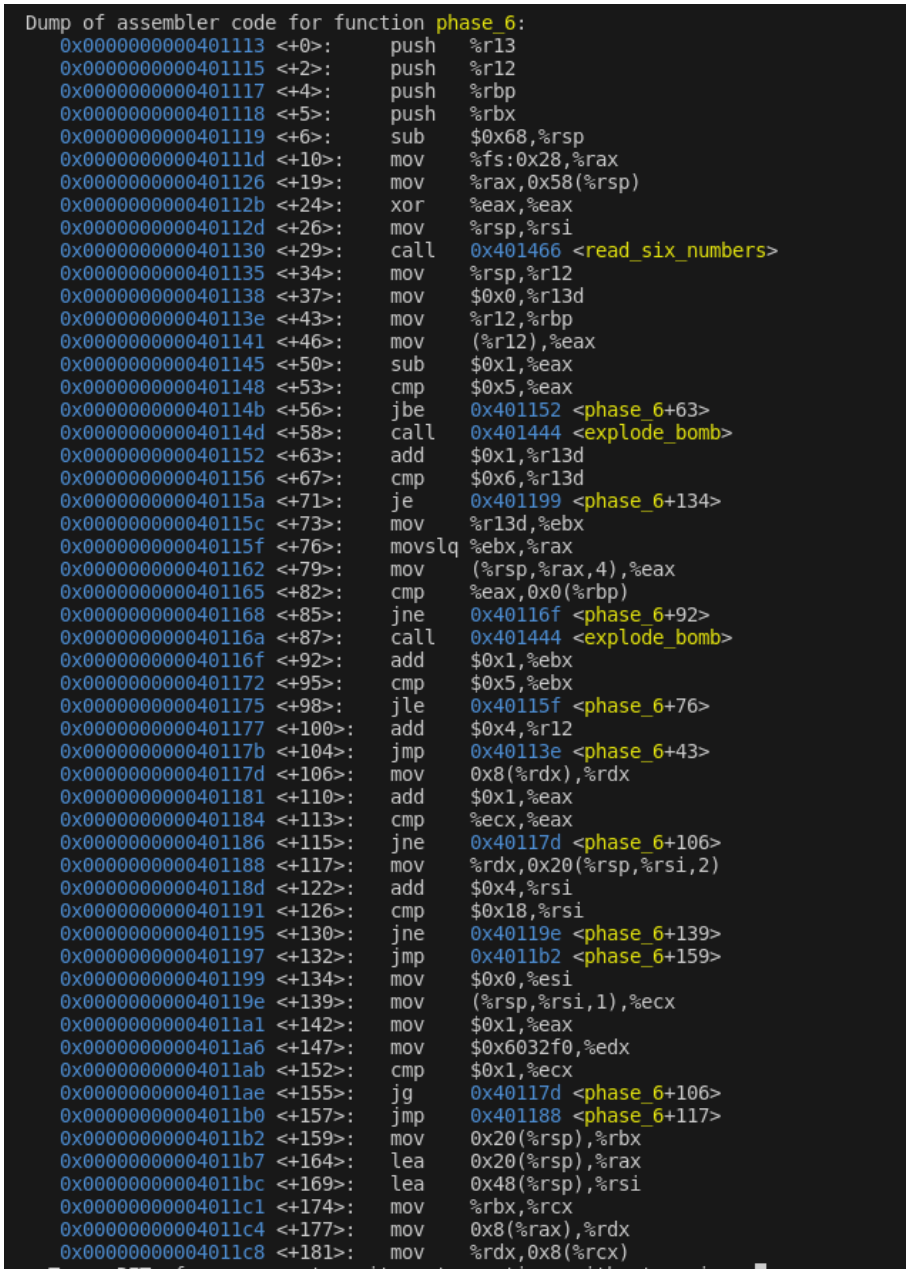
## Phase 5:


Dump of assembler code for function phase_5:
0x0000000000401087 <+0>: sub $0x18,%rsp
0x000000000040108b <+4>: mov %fs:0x28,%rax
0x0000000000401094 <+13>: mov %rax,0x8(%rsp)
0x0000000000401099 <+18>: xor %eax,%eax
0x000000000040109b <+20>: lea 0x4(%rsp),%rcx
0x00000000004010a0 <+25>: mov %rsp,%rdx
0x00000000004010a3 <+28>: mov $0x4025cf,%esi
0x00000000004010a8 <+33>: call 0x400bb0 <__isoc99_sscanf@plt>
0x00000000004010ad <+38>: cmp $0x1,%eax
0x00000000004010b0 <+41>: jg 0x4010b7 <phase_5+48>
0x00000000004010b2 <+43>: call 0x401444 <explode_bomb>
0x00000000004010b7 <+48>: mov (%rsp),%eax
0x00000000004010ba <+51>: and $0xf,%eax
0x00000000004010bd <+54>: mov %eax,(%rsp)
0x00000000004010c0 <+57>: cmp $0xf,%eax
0x00000000004010c3 <+60>: je 0x4010f4 <phase_5+109>
0x00000000004010c5 <+62>: mov $0x0,%ecx
0x00000000004010ca <+67>: mov $0x0,%edx
0x00000000004010cf <+72>: add $0x1,%edx
0x00000000004010d2 <+75>: cltq
0x00000000004010d4 <+77>: mov 0x402480(,%rax,4),%eax
0x00000000004010db <+84>: add %eax,%ecx
0x00000000004010dd <+86>: cmp $0xf,%eax
0x00000000004010e0 <+89>: jne 0x4010cf <phase_5+72>
0x00000000004010e2 <+91>: movl $0xf,(%rsp)
0x00000000004010e9 <+98>: cmp $0xf,%edx
0x00000000004010ec <+101>: jne 0x4010f4 <phase_5+109>
0x00000000004010ee <+103>: cmp 0x4(%rsp),%ecx
0x00000000004010f2 <+107>: je 0x4010f9 <phase_5+114>
0x00000000004010f4 <+109>: call 0x401444 <explode_bomb>
0x00000000004010f9 <+114>: mov 0x8(%rsp),%rax
0x00000000004010fe <+119>: xor %fs:0x28,%rax
0x0000000000401107 <+128>: je 0x40110e <phase_5+135>
0x0000000000401109 <+130>: call 0x400b00 <__stack_chk_fail@plt>
0x000000000040110e <+135>: add $0x18,%rsp
0x0000000000401112 <+139>: ret
End of assembler dump.

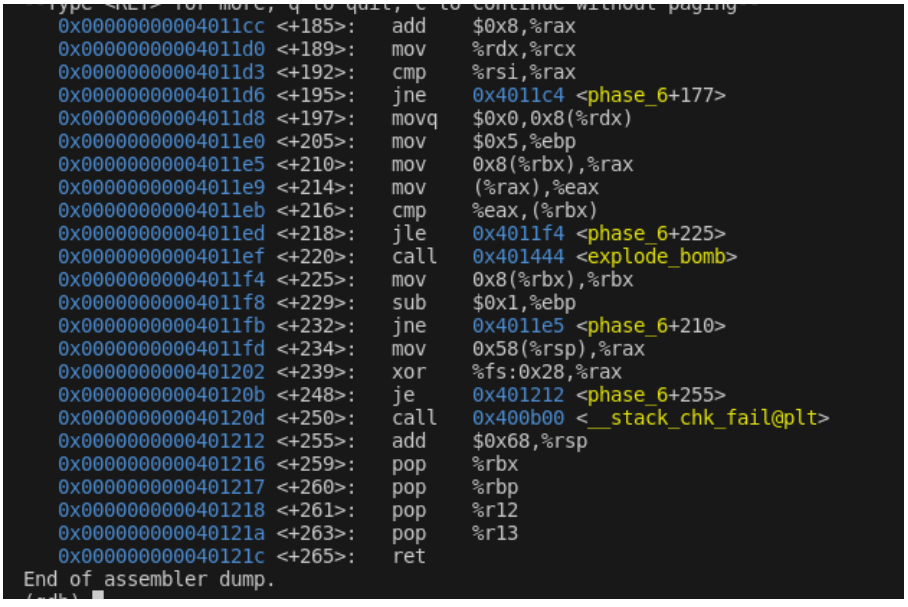
- From the above code we can say that it has a loop.
- By using the ‘x/s 0x4025cf’ command we get to know that we have to give two numbers as input which will be stored at (rsp) and (rsp+4).
- At <+51> taking AND operation of first integer with 0xf and the result is compared with 0xf.
- So the values 15, 31, 63 are prohibited as if the last four bits are 1 the bomb may explode.
- Then at <phase_5+77> the array at 0x402400 is loaded at eax register which gives 15 elements as it runs 15 times.
- So we can get 15 values in array by doing x/15d 0x402400.
- So the sum of all the elements in the array is 115.
- From iteration we get the value 5 which can be taken at 1st integer.

So the fifth phase can be diffused by using the integers:
“5 115”

## Phase 6:



```
Dump of assembler code for function phase_6:
   0x0000000000401113 <+0>:     push   %r13
   0x0000000000401115 <+2>:     push   %r12
   0x0000000000401117 <+4>:     push   %rbp
   0x0000000000401118 <+5>:     push   %rbx
   0x0000000000401119 <+6>:     sub    $0x68,%rsp
   0x000000000040111d <+10>:    mov    %fs:0x28,%rax
   0x0000000000401126 <+19>:    mov    %rax,0x58(%rsp)
   0x000000000040112b <+24>:    xor    %eax,%eax
   0x000000000040112d <+26>:    mov    %rsp,%rsi
   0x0000000000401130 <+29>:    call   0x401466 <read_six_numbers>
   0x0000000000401135 <+34>:    mov    %rsp,%r12
   0x0000000000401138 <+37>:    mov    $0x0,%r13d
   0x000000000040113e <+43>:    mov    %r12,%rbp
   0x0000000000401141 <+46>:    mov    (%r12),%eax
   0x0000000000401145 <+50>:    sub    $0x1,%eax
   0x0000000000401148 <+53>:    cmp    $0x5,%eax
   0x000000000040114b <+56>:    jbe    0x401152 <phase_6+63>
   0x000000000040114d <+58>:    call   0x401444 <explode_bomb>
   0x0000000000401152 <+63>:    add    $0x1,%r13d
   0x0000000000401156 <+67>:    cmp    $0x6,%r13d
   0x000000000040115a <+71>:    je     0x401199 <phase_6+134>
   0x000000000040115c <+73>:    mov    %r13d,%ebx
   0x000000000040115f <+76>:    movslq %ebx,%rax
   0x0000000000401162 <+79>:    mov    (%rsp,%rax,4),%eax
   0x0000000000401165 <+82>:    cmp    %eax,0x0(%rbp)
   0x0000000000401168 <+85>:    jne    0x40116f <phase_6+92>
   0x000000000040116a <+87>:    call   0x401444 <explode_bomb>
   0x000000000040116f <+92>:    add    $0x1,%ebx
   0x0000000000401172 <+95>:    cmp    $0x5,%ebx
   0x0000000000401175 <+98>:    jle    0x40115f <phase_6+76>
   0x0000000000401177 <+100>:   add    $0x4,%r12
   0x000000000040117b <+104>:   jmp    0x40113e <phase_6+43>
   0x000000000040117d <+106>:   mov    0x8(%rdx),%rdx
   0x0000000000401181 <+110>:   add    $0x1,%eax
   0x0000000000401184 <+113>:   cmp    %ecx,%eax
   0x0000000000401186 <+115>:   jne    0x40117d <phase_6+106>
   0x0000000000401188 <+117>:   mov    %rdx,0x20(%rsp,%rsi,2)
   0x000000000040118d <+122>:   add    $0x4,%rsi
   0x0000000000401191 <+126>:   cmp    $0x18,%rsi
   0x0000000000401195 <+130>:   jne    0x40119e <phase_6+139>
   0x0000000000401197 <+132>:   jmp    0x4011b2 <phase_6+159>
   0x0000000000401199 <+134>:   mov    $0x0,%esi
   0x000000000040119e <+139>:   mov    (%rsp,%rsi,1),%ecx
   0x00000000004011a1 <+142>:   mov    $0x1,%eax
   0x00000000004011a6 <+147>:   mov    $0x6032f0,%edx
   0x00000000004011ab <+152>:   cmp    $0x1,%ecx
   0x00000000004011ae <+155>:   jg     0x40117d <phase_6+106>
   0x00000000004011b0 <+157>:   jmp    0x401188 <phase_6+117>
   0x00000000004011b2 <+159>:   mov    0x20(%rsp),%rbx
   0x00000000004011b7 <+164>:   lea    0x20(%rsp),%rax
   0x00000000004011bc <+169>:   lea    0x48(%rsp),%rsi
   0x00000000004011c1 <+174>:   mov    %rbx,%rcx
   0x00000000004011c4 <+177>:   mov    0x8(%rax),%rdx
   0x00000000004011c8 <+181>:   mov    %rdx,0x8(%rcx)
```




- The phase_6 function starts by pushing registers onto the stack and allocating space on the stack by subtracting 0x68 from %rsp.

- The function calls read_six_numbers, which reads six integers from the input and stores them in memory starting from the address in %rsp.
- The function then initializes %r13d to 0 and sets %rbp to the address in %r12, which points to the beginning of the array of numbers.
- It checks the first number in the array (%eax) and ensures it is between 1 and 6 (inclusive). If it's not, the program calls explode_bomb, which triggers a bomb explosion and terminates the program.
- The function loops through the linked list of numbers and checks if each number is equal to the corresponding position in the linked list starting from %rbp. If any number is not equal, explode_bomb is called.
- If all the numbers in the linked list match the inputs given, the function jumps to the end and finishes successfully.

So the sixth phase can be diffused by using the integers:
“2 6 5 1 3 4”