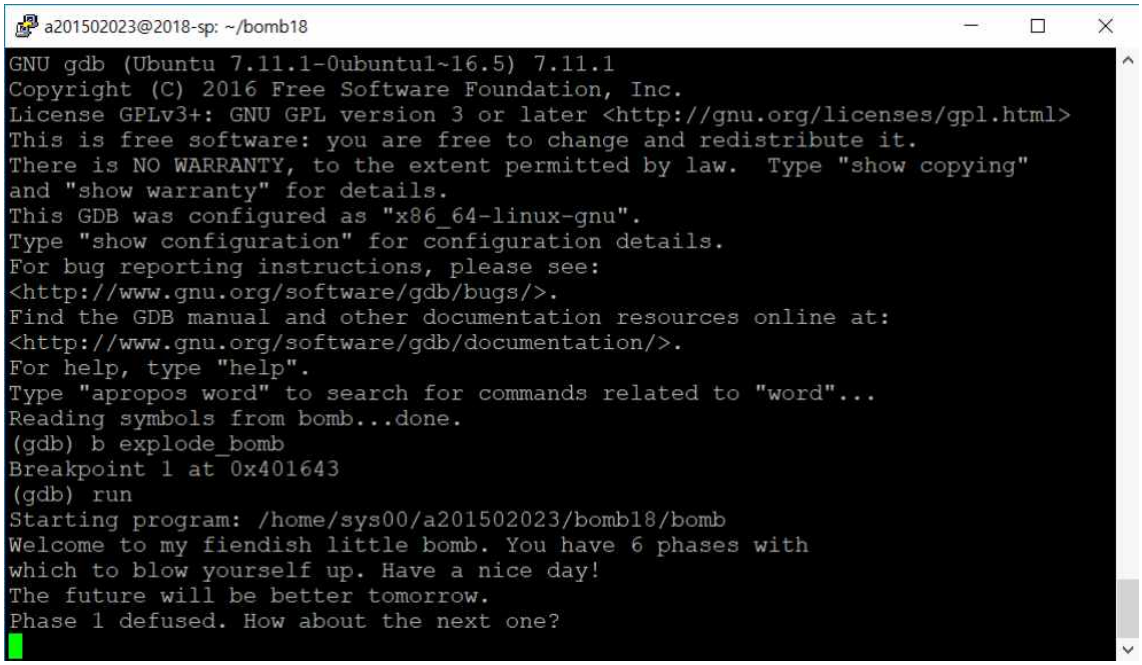
2018 시스템 프로그래밍
- Lab 05 -

제출일자	2018.11.4
분 반	00
이 름	김민기
학 번	201502023

Phase 1 [결과 화면 캡처]



Phase 1 [진행 과정 설명]

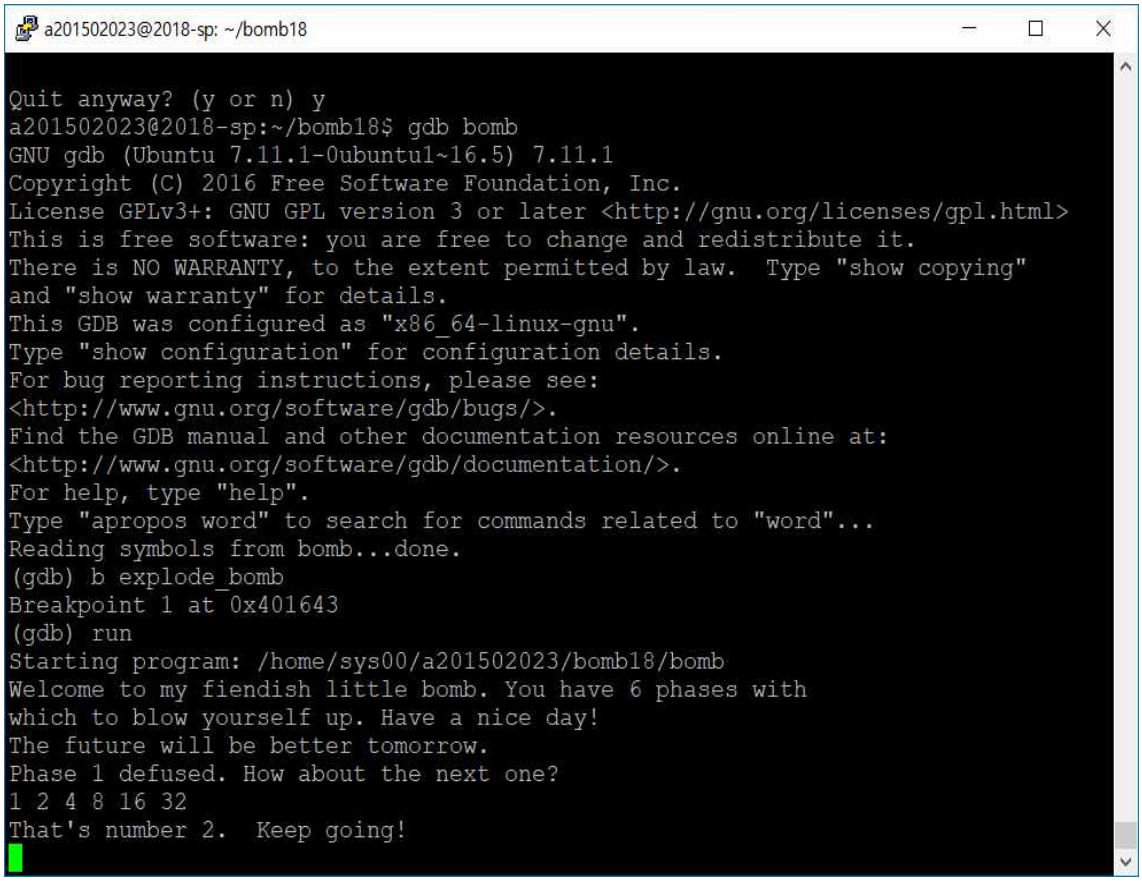
- 해당 단계와 explode_bomb에 break를 걸고 disas로 해당단계의 내용을 파악.
- <strings_not_equal> 전에 있는 move $0x402610,%esi는 $0x402610안에 있는 값을 %esi에 넣는 것이므로 x/s 0x402610을 이용해 값을 확인한다.
- 바로 뒤에 <strings_not_equal>을 보고 확인한 값과 비교해서 두 값이 다르면 bomb가 터지고 같으면 터지지 않는다.

```
Dump of assembler code for function phase_1:
=> 0x0000000000400f2d <+0>:     sub    $0x8,%rsp
   0x0000000000400f31 <+4>:     mov    $0x402610,%esi
   0x0000000000400f36 <+9>:     callq  0x40136f <strings_not_equal>
   0x0000000000400f3b <+14>:    test   %eax,%eax
   0x0000000000400f3d <+16>:    je     0x400f44 <phase_1+23>
   0x0000000000400f3f <+18>:    callq  0x401643 <explode_bomb>
   0x0000000000400f44 <+23>:    add    $0x8,%rsp
   0x0000000000400f48 <+27>:    retq
End of assembler dump.
```

Phase 1 [정답]

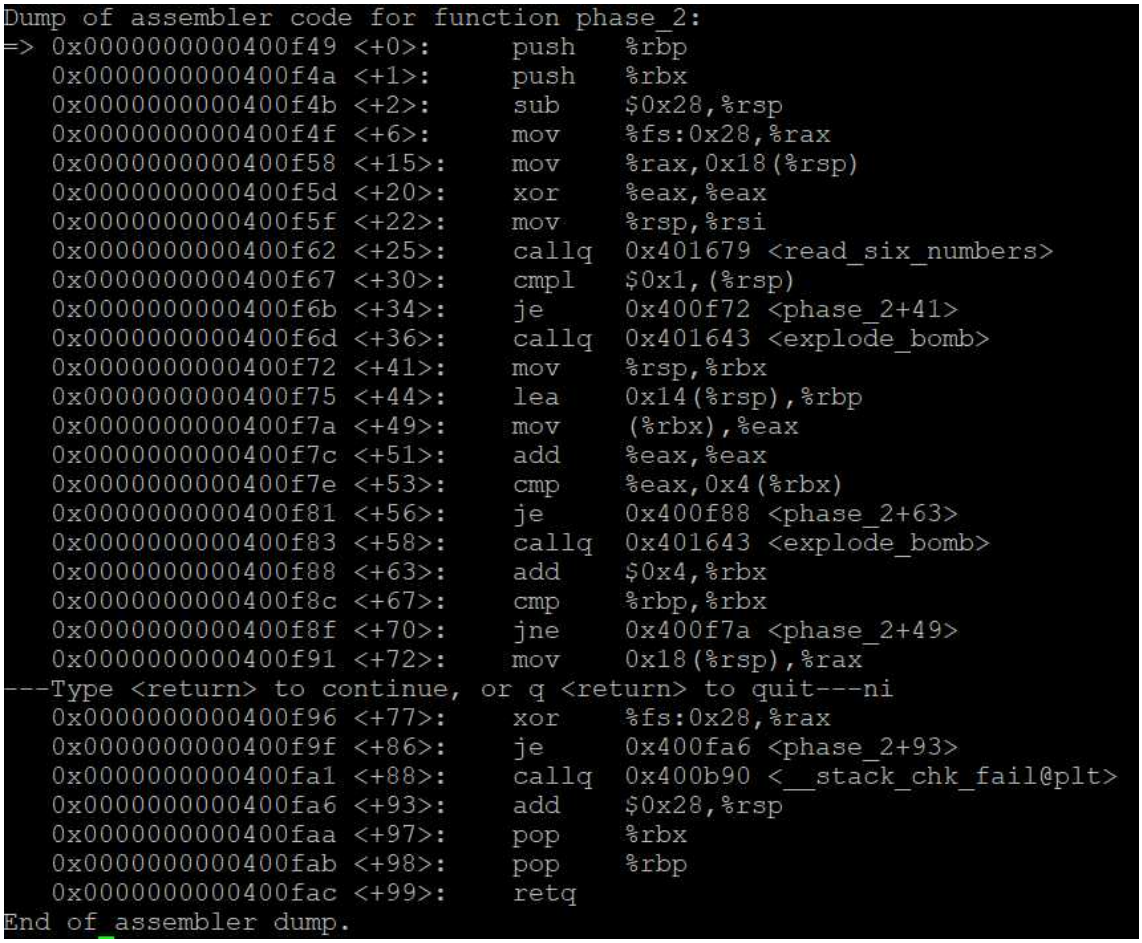
The future will be better tomorrow.

Phase 2 [결과 화면 캡처]



Phase 2 [진행 과정 설명]

- 해당 단계와 explode_bomb에 break를 걸고 disas로 해당단계의 내용을 파악.
- <+25> read_six_number를 보면 숫자 6개를 입력하라는 것을 알 수 있다.
- display $rax를 통해 변화과정을 살펴서 until *0x400f88(폭탄 전)로 rax값을 볼 수 있다.
- 1 2 3 4 5 6을 입력했을 때 2까지는 무사히 지나가고 다음 rax = 4를 요구할 때 폭탄을 호출하는 것으로 보아 3번째 값이 4임을 알 수 있다. 1 2 4 5 6 7로 다시 입력하고 위 과정을 반복하면 4까지는 지나가고 그 다음에 rax = 8을 요구하는 것으로 보아 1 2 4 8 …이라는 것을 알 수 있다. 이렇게 반복하면 정답을 찾아낼 수 있다.

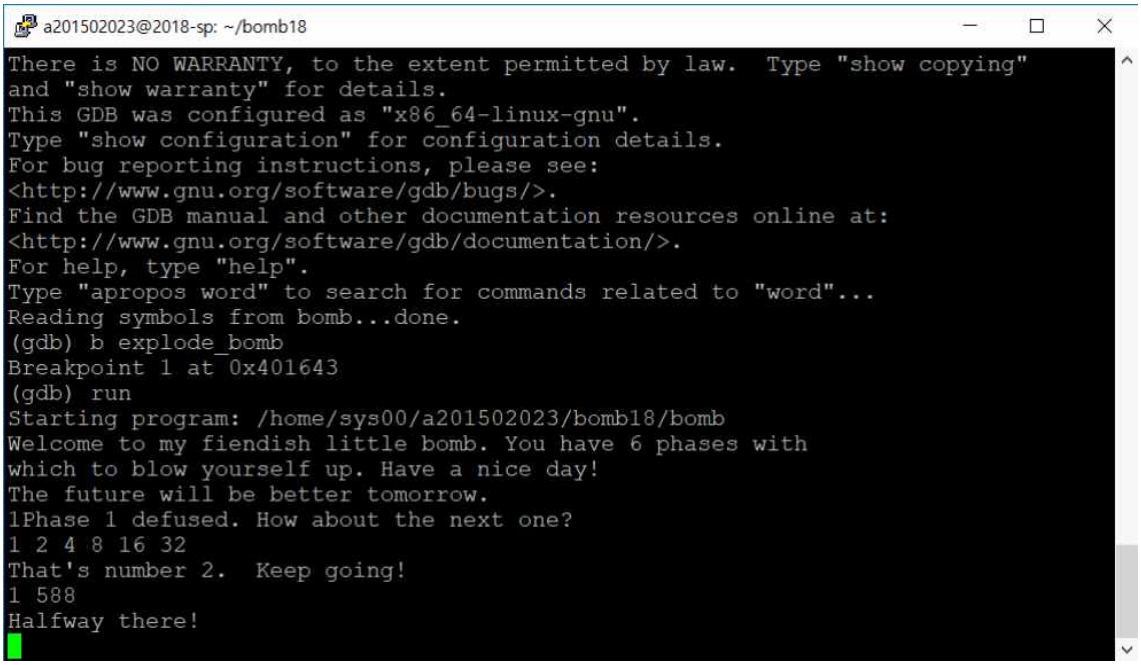


```
Dump of assembler code for function phase_2:
=> 0x0000000000400f49 <+0>:     push   %rbp
   0x0000000000400f4a <+1>:     push   %rbx
   0x0000000000400f4b <+2>:     sub    $0x28,%rsp
   0x0000000000400f4f <+6>:     mov    %fs:0x28,%rax
   0x0000000000400f58 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400f5d <+20>:    xor    %eax,%eax
   0x0000000000400f5f <+22>:    mov    %rsp,%rsi
   0x0000000000400f62 <+25>:    callq  0x401679 <read_six_numbers>
   0x0000000000400f67 <+30>:    cmpl   $0x1,(%rsp)
   0x0000000000400f6b <+34>:    je     0x400f72 <phase_2+41>
   0x0000000000400f6d <+36>:    callq  0x401643 <explode_bomb>
   0x0000000000400f72 <+41>:    mov    %rsp,%rbx
   0x0000000000400f75 <+44>:    lea    0x14(%rsp),%rbp
   0x0000000000400f7a <+49>:    mov    (%rbx),%eax
   0x0000000000400f7c <+51>:    add    %eax,%eax
   0x0000000000400f7e <+53>:    cmp    %eax,0x4(%rbx)
   0x0000000000400f81 <+56>:    je     0x400f88 <phase_2+63>
   0x0000000000400f83 <+58>:    callq  0x401643 <explode_bomb>
   0x0000000000400f88 <+63>:    add    $0x4,%rbx
   0x0000000000400f8c <+67>:    cmp    %rbp,%rbx
   0x0000000000400f8f <+70>:    jne    0x400f7a <phase_2+49>
   0x0000000000400f91 <+72>:    mov    0x18(%rsp),%rax
---Type <return> to continue, or q <return> to quit---ni
   0x0000000000400f96 <+77>:    xor    %fs:0x28,%rax
   0x0000000000400f9f <+86>:    je     0x400fa6 <phase_2+93>
   0x0000000000400fa1 <+88>:    callq  0x400b90 <__stack_chk_fail@plt>
   0x0000000000400fa6 <+93>:    add    $0x28,%rsp
   0x0000000000400faa <+97>:    pop    %rbx
   0x0000000000400fab <+98>:    pop    %rbp
   0x0000000000400fac <+99>:    retq
End of assembler dump.
```

Phase2 [정답]

1 2 4 8 16 32

Phase 3 [결과 화면 캡처]



Phase 3 [진행 과정 설명]

- 해당 단계와 explode_bomb에 break를 걸고 disas로 해당단계의 내용을 파악.
- That's number 2. Keep going! 이 문구를 보아 두 가지의 숫자를 맞춰야 하는 것으로 보인다.
- <+38>에서 처음으로 비교를 하게 되는데 eax와 1일 비교하여 결과가 크면 분기를 하는데, 여기서 알 수 있는 것은 eax는 첫 숫자이며 1보다 커야 한다는 것이다.
- <+48>로 가서 7이랑 비교를 한다. 7보다 크면 폭탄의 주소로 이동하기 때문에 첫 번째 숫자가 7보다는 작아야 한다는 것을 알 수 있다.
- <+54>에서 첫 번째 값을 eax로 옮기고 eax의 값에 따라 점프를 하게 되는데 <+57>의 *0x402660(,%rax,8)는 무조건 점프를 뜻한다. phase_3+130부분은 두 번째의 입력갑과 eax를 비교해서 같지 않으면 폭탄이 터진다는 뜻이다.
- eax는 7가지의 경우가 존재하는데, 1을 넣었을 경우 첫 부분에 도달해서 <+64>를 보면 eax에 0x24c가 들어가는 것을 알 수 있다. 0x24c는 16진수이므로 10진수로 변환하면 (16*16*2 + 16*4 + 12) = 588이 나온다. 이 단계는 7가지의 정답이 있지만 1을 입력했을 때의 정답은 588이다.

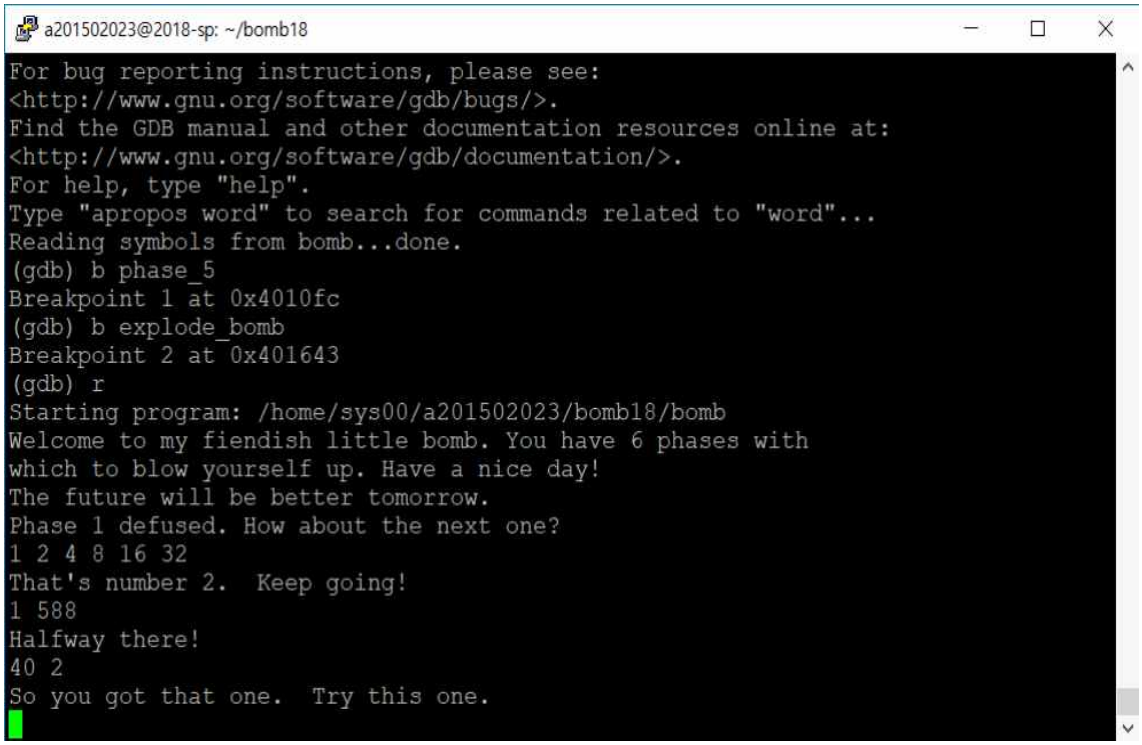
```
Dump of assembler code for function phase_3:
=> 0x0000000000400fad <+0>:     sub    $0x18,%rsp
   0x0000000000400fb1 <+4>:     mov    %fs:0x28,%rax
   0x0000000000400fba <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000400fbf <+18>:    xor    %eax,%eax
   0x0000000000400fc1 <+20>:    lea    0x4(%rsp),%rcx
   0x0000000000400fc6 <+25>:    mov    %rsp,%rdx
   0x0000000000400fc9 <+28>:    mov    $0x40292d,%esi
   0x0000000000400fce <+33>:    callq  0x400c40 <__isoc99_sscanf@plt>
   0x0000000000400fd3 <+38>:    cmp    $0x1,%eax
   0x0000000000400fd6 <+41>:    jg     0x400fdd <phase_3+48>
   0x0000000000400fd8 <+43>:    callq  0x401643 <explode_bomb>
   0x0000000000400fdd <+48>:    cmpl   $0x7,(%rsp)
   0x0000000000400fe1 <+52>:    ja     0x40101e <phase_3+113>
   0x0000000000400fe3 <+54>:    mov    (%rsp),%eax
   0x0000000000400fe6 <+57>:    jmpq   *0x402660(,%rax,8)
   0x0000000000400fed <+64>:    mov    $0x24c,%eax
   0x0000000000400ff2 <+69>:    jmp    0x40102f <phase_3+130>
   0x0000000000400ff4 <+71>:    mov    $0x21c,%eax
   0x0000000000400ff9 <+76>:    jmp    0x40102f <phase_3+130>
   0x0000000000400ffb <+78>:    mov    $0xde,%eax
   0x0000000000401000 <+83>:    jmp    0x40102f <phase_3+130>
   0x0000000000401002 <+85>:    mov    $0x3e7,%eax
   0x0000000000401007 <+90>:    jmp    0x40102f <phase_3+130>
   0x0000000000401009 <+92>:    mov    $0x95,%eax
   0x000000000040100e <+97>:    jmp    0x40102f <phase_3+130>
   0x0000000000401010 <+99>:    mov    $0x1bc,%eax
   0x0000000000401015 <+104>:   jmp    0x40102f <phase_3+130>
   0x0000000000401017 <+106>:   mov    $0xb8,%eax
   0x000000000040101c <+111>:   jmp    0x40102f <phase_3+130>
   0x000000000040101e <+113>:   callq  0x401643 <explode_bomb>
   0x0000000000401023 <+118>:   mov    $0x0,%eax
   0x0000000000401028 <+123>:   jmp    0x40102f <phase_3+130>
```

```
   0x000000000040102a <+125>:   mov    $0x277,%eax
   0x000000000040102f <+130>:   cmp    0x4(%rsp),%eax
   0x0000000000401033 <+134>:   je     0x40103a <phase_3+141>
   0x0000000000401035 <+136>:   callq  0x401643 <explode_bomb>
   0x000000000040103a <+141>:   mov    0x8(%rsp),%rax
   0x000000000040103f <+146>:   xor    %fs:0x28,%rax
   0x0000000000401048 <+155>:   je     0x40104f <phase_3+162>
   0x000000000040104a <+157>:   callq  0x400b90 <__stack_chk_fail@plt>
   0x000000000040104f <+162>:   add    $0x18,%rsp
   0x0000000000401053 <+166>:   retq
End of assembler dump.
```

Phase 3 [정답]

1 588

Phase 4 [결과 화면 캡처]



Phase 4 [진행 과정 설명]

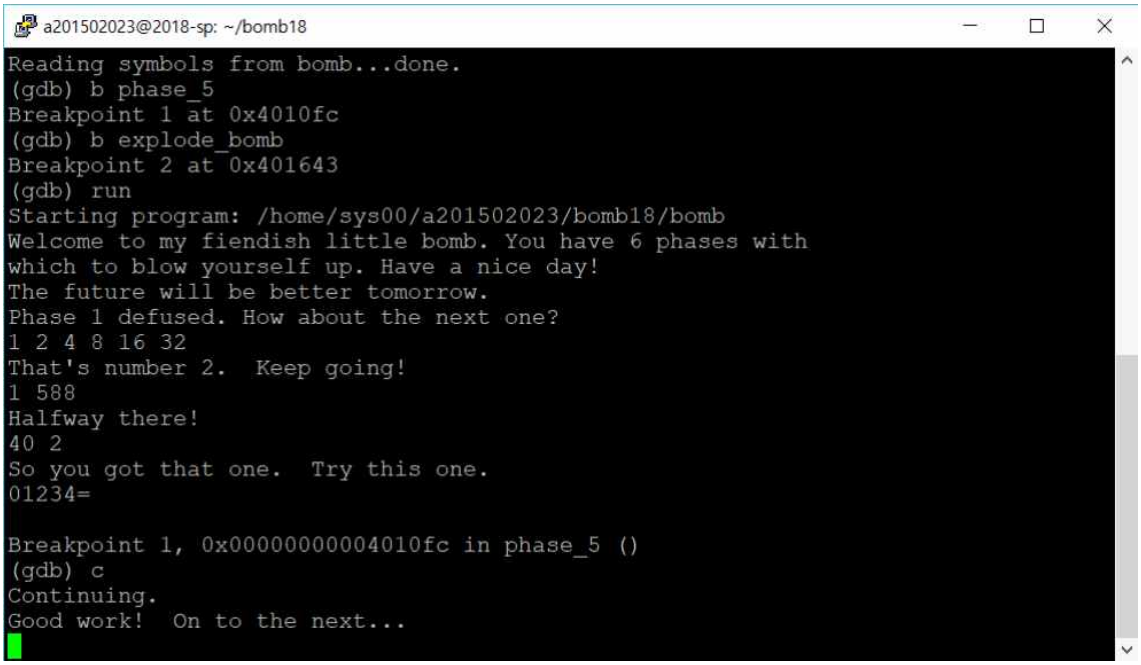
- 해당 단계와 explode_bomb에 break를 걸고 disas로 해당단계의 내용을 파악.
- cmp로 2와 비교하는 부분이 있어서 2 5를 입력하고 실행해보았다. ni로 넘어가면서 <+43, +46, +49>를 보니 rsp값을 eax에 옮기고 eax에서 2를 빼서 2와 비교해, 그 값이 2보다 작거나 같아야 하는 구조이다. (eax - 2 <= 2).
- 여기서 eax는 두 번째 숫자를 말하므로, 두 번째 입력한 숫자는 2, 3, 4의 세 가지 경우가 된다. 즉 두 번째 입력 값에 따라 첫 번째 값이 결정되는 것이다.
- 즉 fun4함수를 실행한 뒤 레지스터의 값을 확인하면되는데 <+72>줄을 i r에서 rax의 값을 보면 0x28이다. 0x28은 16진수이므로 10진수로 변경하면 40이 된다.
- 따라서 정답은 40, 2 이다.

```
Breakpoint 4, 0x000000000040108f in phase_4 ()
(gdb) disas
Dump of assembler code for function phase_4:
=> 0x000000000040108f <+0>:     sub    $0x18,%rsp
   0x0000000000401093 <+4>:     mov    %fs:0x28,%rax
   0x000000000040109c <+13>:    mov    %rax,0x8(%rsp)
   0x00000000004010a1 <+18>:    xor    %eax,%eax
   0x00000000004010a3 <+20>:    mov    %rsp,%rcx
   0x00000000004010a6 <+23>:    lea    0x4(%rsp),%rdx
   0x00000000004010ab <+28>:    mov    $0x40292d,%esi
   0x00000000004010b0 <+33>:    callq  0x400c40 <__isoc99_sscanf@plt>
   0x00000000004010b5 <+38>:    cmp    $0x2,%eax
   0x00000000004010b8 <+41>:    jne    0x4010c5 <phase_4+54>
   0x00000000004010ba <+43>:    mov    (%rsp),%eax
   0x00000000004010bd <+46>:    sub    $0x2,%eax
   0x00000000004010c0 <+49>:    cmp    $0x2,%eax
   0x00000000004010c3 <+52>:    jbe    0x4010ca <phase_4+59>
   0x00000000004010c5 <+54>:    callq  0x401643 <explode_bomb>
   0x00000000004010ca <+59>:    mov    (%rsp),%esi
   0x00000000004010cd <+62>:    mov    $0x6,%edi
   0x00000000004010d2 <+67>:    callq  0x401054 <func4>
   0x00000000004010d7 <+72>:    cmp    0x4(%rsp),%eax
   0x00000000004010db <+76>:    je     0x4010e2 <phase_4+83>
   0x00000000004010dd <+78>:    callq  0x401643 <explode_bomb>
   0x00000000004010e2 <+83>:    mov    0x8(%rsp),%rax
   0x00000000004010e7 <+88>:    xor    %fs:0x28,%rax
   0x00000000004010f0 <+97>:    je     0x4010f7 <phase_4+104>
   0x00000000004010f2 <+99>:    callq  0x400b90 <__stack_chk_fail@plt>
   0x00000000004010f7 <+104>:   add    $0x18,%rsp
   0x00000000004010fb <+108>:   retq
End of assembler dump.
```

Phase 4 [정답]

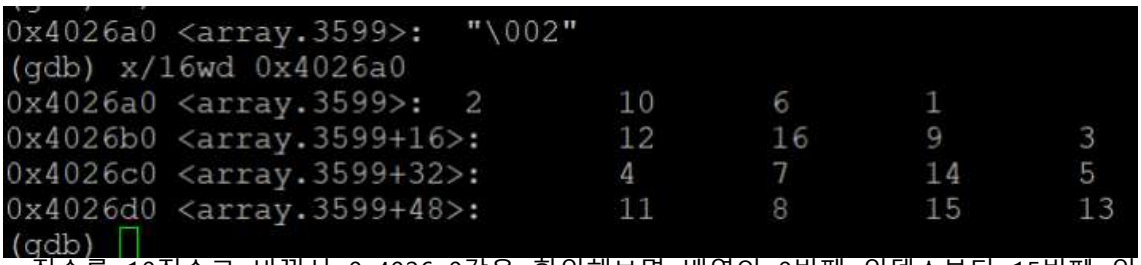
40 2

Phase 5 [결과 화면 캡처]



Phase 5 [진행 과정 설명]

- 해당 단계와 explode_bomb에 break를 걸고 disas로 해당단계의 내용을 파악.
- <+15>의 string_length와 <+12>의 $0x6, %eax로 보아 6개의 문자열을 입력하는 것을 알 수 있다.
- <+37>을 보면 0x4026a0+(4*%rdx)값을 %ecx로 옮겨주는데 x/16wd명령어를 사용해 16



진수를 10진수로 바꿔서 0x4026a0값을 확인해보면 배열의 0번째 인덱스부터 15번째 인덱스까지 10진수로 값이 들어가 있는 것을 볼 수 있다.

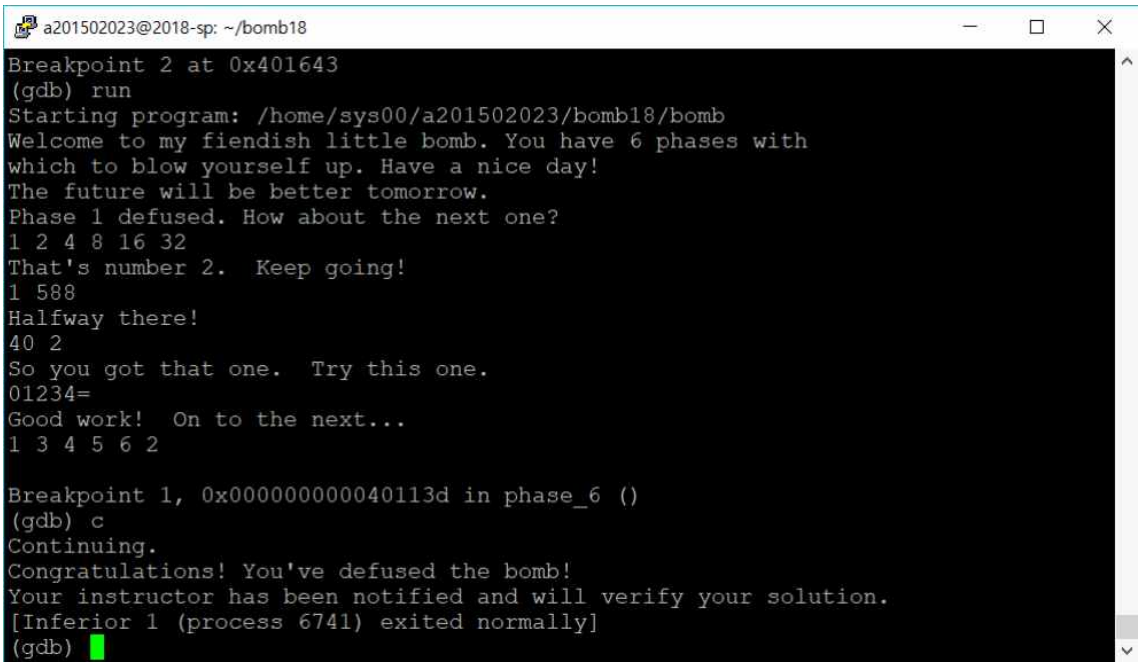
- <+53>를 에 ecx와 $0x27와 비교하는데, 0x27은 10진수로 변환하면 39이므로 위 배열 값 중 6개를 뽑아 더한 값이 39가 나오는 인덱스를 뽑아준다.
- 2 + 10 + 6 + 1 + 12 + 8 = 39 -> 인덱스 값은 0 1 2 3 4 d이다. 아스키코드 값을 보니 0 1 2 3 4 = 이었고 5단계의 정답이다.

```
Breakpoint 5, 0x00000000004010fc in phase_5 ()
(gdb) disas
Dump of assembler code for function phase_5:
=> 0x00000000004010fc <+0>:     push   %rbx
   0x00000000004010fd <+1>:     mov    %rdi,%rbx
   0x0000000000401100 <+4>:     callq  0x401351 <string_length>
   0x0000000000401105 <+9>:     cmp    $0x6,%eax
   0x0000000000401108 <+12>:    je     0x40110f <phase_5+19>
   0x000000000040110a <+14>:    callq  0x401643 <explode_bomb>
   0x000000000040110f <+19>:    mov    %rbx,%rax
   0x0000000000401112 <+22>:    lea    0x6(%rbx),%rdi
   0x0000000000401116 <+26>:    mov    $0x0,%ecx
   0x000000000040111b <+31>:    movzbl (%rax),%edx
   0x000000000040111e <+34>:    and    $0xf,%edx
   0x0000000000401121 <+37>:    add    0x4026a0(,%rdx,4),%ecx
   0x0000000000401128 <+44>:    add    $0x1,%rax
   0x000000000040112c <+48>:    cmp    %rdi,%rax
   0x000000000040112f <+51>:    jne    0x40111b <phase_5+31>
   0x0000000000401131 <+53>:    cmp    $0x27,%ecx
   0x0000000000401134 <+56>:    je     0x40113b <phase_5+63>
   0x0000000000401136 <+58>:    callq  0x401643 <explode_bomb>
   0x000000000040113b <+63>:    pop    %rbx
   0x000000000040113c <+64>:    retq
End of assembler dump.
```

Phase 5 [정답]

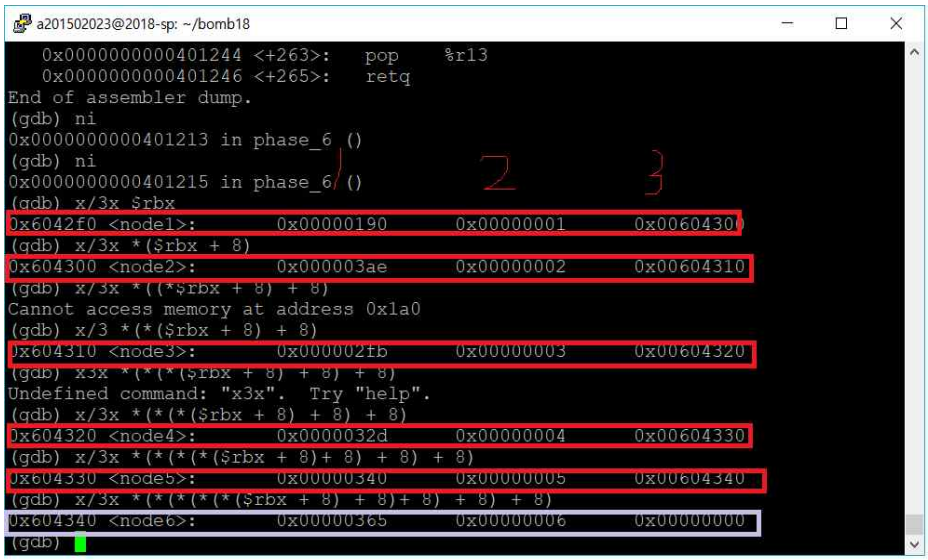
0 1 2 3 4 =

Phase 6 [결과 화면 캡처]



Phase 6 [진행 과정 설명]

- 해당 단계와 explode_bomb에 break를 걸고 disas로 해당단계의 내용을 파악.
- <+29>에 read_six_number를 보니 6개의 숫자를 입력해야하는 것을 알 수 있다.



- <+210 ~ +218>을 보고 %rbx, %rbx + 0x8, %rbx + 0x10...의 내용을 살펴보면 <node1

- ~ node6>을 볼 수 있다. 1번은 노드의 값이고 2번은 목록의 노드 위치이며 3번은 목록의 다음 노드에 대한 포인터이다..
- <+216, +218>을 보고 생각해보면 다음 노드의 값이 현재 노드의 값보다 작으면 폭탄이 폭발한다는 것을 알 수 있다. 즉 노드를 오름차순으로 정렬해야 한다.
- (1) 0x00000190, (2) 0x000003ae, (3) 0x000002fb, (4) 0x0000032d, (5) 0x00000340, (6) 0x00000365, 오름차순으로 정렬하면 1 3 4 5 6 2 순이다.

Phase 6	[정답]

1 3 4 5 6 2