Lab # 2 . 20210354 김상협

<Problem 2-1>

```
0x0000000000401136 <+0>:     sub    $0x28,%rsp
0x000000000040113a <+4>:     mov    $0x402004,%edi
0x000000000040113f <+9>:     callq  0x401030 <puts@plt>
0x0000000000401144 <+14>:    mov    %rsp,%rsi
0x0000000000401147 <+17>:    mov    $0x402018,%edi
0x000000000040114c <+22>:    mov    $0x0,%eax
0x0000000000401151 <+27>:    callq  0x401040 <__isoc99_scanf@plt>
```

이 부분은 함수 시작 시 스택을 확보하고 puts() 함수를 호출합니다. 그 다음 부분은 c코드를 보고 유추했는데, scanf("%31s", buf); 를 보고 추론했습니다. mov %rsp, %rsi 를 사용해 현재 스택의 주소를 넘길 수 있도록 하였고 edi 에는 "%31s" 를 나타내는 주소 값이 들어가 있었다고 생각했습니다.

```
0x0000000000401156 <+32>:    mov    $0x0,%eax
0x000000000040115b <+37>:    movslq %eax,%rdx
0x000000000040115e <+40>:    movzbl 0x404038(%rdx),%edx
0x0000000000401165 <+47>:    test   %dl,%dl
```

scanf 이후 eax 를 다시 0으로 초기화 시켜준 뒤 movslq 를 통해서 eax 값을 rdx 로 넘겨주고 넘겨준 rdx 값을 통해서 0x404038 주소에서 rdx 만큼 주소를 읽은 뒤에 해당하는 값을 edx (dl) 로 넘기게 됩니다.

저는 이부분이 problem 1 번에서 가장 중요한 분기점이라고 생각했습니다. 일단 여기까지 코드를 봐서는 정확히 알 수 없었지만 0x404038 주소에 있는 값이 해당 어셈블리 코드에서 중요하게 작동하고 있고, rdx 값이 계속 변하면서 해당 주소와 상호작용을 하면서 코드가 돌아갈 것으로 추측했고, 조금 더 자세하게 뒤에서 설명하도록 하겠습니다.

```
0x0000000000401167 <+49>:    je     0x401176 <main+64>
0x0000000000401169 <+51>:    movslq %eax,%rcx
0x000000000040116c <+54>:    cmp    %dl,(%rsp,%rcx,1)
0x000000000040116f <+57>:    jne    0x401193 <main+93>
0x0000000000401171 <+59>:    add    $0x1,%eax
0x0000000000401174 <+62>:    jmp    0x40115b <main+37>
0x0000000000401176 <+64>:    mov    $0x1,%eax
0x000000000040117b <+69>:    test   %eax,%eax
0x000000000040117d <+71>:    je     0x40119a <main+100>
0x0000000000401198 <+98>:    jmp    0x40117b <main+69>
0x000000000040119a <+100>:   mov    $0x402038,%edi
0x000000000040119f <+105>:   callq  0x401030 <puts@plt>
0x00000000004011a4 <+110>:   jmp    0x401189 <main+83>
```

dl 값이 0인 경우 : <main 64>부분으로 jump 하게 되고 64 에서는 eax 값을 1로 변경합니다. 이때 eax 가 1 이므로 <main> 100 으로 jump 하지 않고 73번째 줄로 가서 edi 에 $0x40201d 주소값을 넣고 put 함수를 호출합니다. 이후에 eax를 다시 0으로 설정한뒤 스택을 되돌리고 ret 합니다.

DI 값이 0이 아닌경우 :

0x0000000000401169 <+51>: movslq %eax,%rcx ; rcx = (long)i 0x000000000040116c <+54>: cmp %dl,(%rsp,%rcx,1) 0x000000000040116f <+57>: jne 0x401193 <main+93>

0 이 아니라면 <main 64>로 jump 하지 않고, <+51> 줄에 movslq 을 통해서 eax 값을 rcx 로 옮겨준 뒤 cmp %dl,(%rsp,%rcx,1) 을 통해서 두 값을 비교합니다. 이 비교문으로 처음 했던 예측을 더 구체적으로 추론했습니다. 일단 rsp+rcx 이므로 rsp 는 저희가 아까 scanf 입력으로 받아서 스택안에 넣었던 문자열이고, 추가로 rcx 를 더할 때 byte 단위가 1인것으로 미루어보아 문자열 배열을 비교하고 있다고 추론했습니다. 다시 돌아가서 해당 문자열(인덱스와)과 같지 않은 경우는 jne 를 통해서 <main 93> 으로 이동합니다. <Main 93> 에서는 eax 값을 0으로 설정한 뒤 main 69 번으로 jump 합니다. 이후에 <main 69,70> 을 통해서 dl 이 0인 경우와 다르게 <main 100> 으로 jump 합니다. (이때 dl 값이 0일때가 최종 성공 분기임을 예측했습니다. -> 실패 분기와 다르기 때문) 그 이후로는 위에 설명한 방법과 유사하게 함수를 마무리합니다.

해당 문자열 (인덱스) 와 같은 경우 에는 <main 57> 에서 jmp 하지 않습니다. 이 경우에는 기존 eax(인덱스값) 을 +1 해준뒤에 <main 37>로 돌아갑니다. 그 뒤에는 다시 위의 루프를 다시 돌립니다. 따라서 저는 최종적으로 입력값과 인덱스를 하나씩 변경해서 그 문자열과 일치 여부를 따진다고 추론했습니다. 그리고 그 문자열은 0x404038 주소에 저장되어 있을 것입니다. 따라서 0x404038 에 해당하는 값을 확인하였고

```
(gdb) x/s 0x404038
0x404038 <msg>: "CSE3030@Sogang"
```

```
def solve():
    prog = Program("./problem1.bin")
    print(prog.read_line()) # Read the initial message of the program.
    prog.send_line("CSE3030@Sogang") # Send your input to the program.
    print(prog.read_line()) # Read the response from the program.
```

```
cse20210354@cspro2:~/컴시개/Lab2/2-1$ ./problem1.bin
Provide your input:
CSE3030@Sogang
You passed the challenge!
```

정답을 입력해보니 다음과 같은 결과를 얻었습니다.

Problem 2.2

```
0x0000000000401144 <+14>:    lea    0x4(%rsp),%rcx
0x0000000000401149 <+19>:    lea    0x8(%rsp),%rdx
0x000000000040114e <+24>:    lea    0xc(%rsp),%rsi
0x0000000000401153 <+29>:    mov    $0x402018,%edi
0x0000000000401158 <+34>:    mov    $0x0,%eax
0x000000000040115d <+39>:    callq  0x401040 <__isoc99_scanf@plt>
```

이전문제와 비슷하나 problem 2.2 에서는 입력값을 3개를 받습니다. Problem2.c 파일을 보면 scanf에서 x,y,z 이렇게 세개의 변수를 입력 받습니다. %rsi = &x , %rdx= &y , %rcx=&z 이렇게 입력 값을 받습니다.

(이제부터 편의상 x , y , z 라고 써서 설명하겠습니다.)

```
0x0000000000401162 <+44>:    mov    0xc(%rsp),%edx
0x0000000000401166 <+48>:    cmp    $0x40,%edx
0x0000000000401169 <+51>:    jle    0x4011be <main+136>
0x000000000040116b <+53>:    mov    $0x1,%ecx
0x0000000000401170 <+58>:    cmp    $0x60,%edx
0x0000000000401173 <+61>:    jle    0x40117a <main+68>
0x0000000000401175 <+63>:    mov    $0x0,%ecx
```

Z 값을 edx 레지스터에 옮깁니다. 이후에 edx 이 0x40(64) 초과 0x60(96)이하인 경우에는 ecx 를 1로 설정하고 이외의 범위에서는 0이 됩니다.

```
0x000000000040117a <+68>:    mov    0x4(%rsp),%eax
0x000000000040117e <+72>:    cmp    $0x200,%eax
0x0000000000401183 <+77>:    jg     0x40118a <main+84>
0x0000000000401185 <+79>:    mov    $0x0,%ecx
0x000000000040118a <+84>:    cmp    $0x230,%eax
0x000000000040118f <+89>:    jle    0x401196 <main+96>
0x0000000000401191 <+91>:    mov    $0x0,%ecx
```

<main 68> 에서는 0x4(%rsp)(x값) 값을 eax 에 mov 한 뒤에 해당 값을 0x200(512) 와 비교해서 512 초과인 경우와 0x230(560) 이하인 경우에만 ecx 상태를 앞에서 설정한 것과 같게 유지하고 아닌 경우에는 ecx 를 0으로 설정합니다.

```
0x0000000000401196 <+96>:    mov    0x8(%rsp),%esi
ype <RET> for more, q to quit, c to continue without paging--
0x000000000040119a <+100>:   mov    %esi,%edi
0x000000000040119c <+102>:   sub    %edx,%edi
0x000000000040119e <+104>:   sub    %esi,%eax
0x00000000004011a0 <+106>:   add    %eax,%eax
0x00000000004011a2 <+108>:   cmp    %eax,%edi
0x00000000004011a4 <+110>:   jne    0x4011aa <main+116>
0x00000000004011a6 <+112>:   test   %ecx,%ecx
0x00000000004011a8 <+114>:   jne    0x4011c5 <main+143>
0x00000000004011aa <+116>:   mov    $0x402040,%edi
0x00000000004011af <+121>:   callq  0x401030 <puts@plt>
0x00000000004011b4 <+126>:   mov    $0x0,%eax
0x00000000004011b9 <+131>:   add    $0x18,%rsp
0x00000000004011bd <+135>:   retq
0x00000000004011be <+136>:   mov    $0x0,%ecx
0x00000000004011c3 <+141>:   jmp    0x401170 <main+58>
0x00000000004011c5 <+143>:   mov    $0x402021,%edi
0x00000000004011ca <+148>:   callq  0x401030 <puts@plt>
0x00000000004011cf <+153>:   jmp    0x4011b4 <main+126>
```

그 다음으로는 이제 결과값들을 연산해줍니다. 0x8(%rsp)(y) 값을 esi 레지스터에 옮긴 뒤, edi 에 다시 y 값을 mov 해줍니다. 그 다음 sub %edx , %edi 연산을 통해서 edi = y – z 값을 가지게 됩니다. 다음 줄에서는 sub %esi, %eax 연산을 통해서 eax = x-y 값을 가지게 됩니다. 이후 add 연산을 통해 %eax = 2(%eax) 즉 2(x-y) 값을 가지게 됩니다. 이후 cmp %eax , %edi 값의 비교를 통해 둘이 같지 않다면 <main 116> 으로 jmp 하게 되고 만약 둘이 일치한다면 무시합니다.

먼저 둘이 같다는 전제로 먼저 설명을 하겠습니다. 둘의 값이 같은 경우에서 ecx 가 1인 경우에는 <main 143> 으로 jmp 합니다. (같아도 ecx 가 0인 경우에는 jmp x) 따라서 이 구조를 살펴보면 ecx 가 1인 조건을 만족할 뿐만 아니라 y-z = 2(x-y) 식을 만족해야 한다는 것을 알 수 있습니다. 이 조건을 모두 만족한 경우 <main 143>으로 jmp 해서 성공했다는 메시지를 출력하고 다시 스택을 복구한 뒤 ret 을 해주게 됩니다. 만약 같지 않은 경우에는 <main 116> 으로 jmp하고 다른 메시지를 출력 후 그대로 스택을 복구한 뒤 ret 을 해줍니다. 정리해보면 (z 값이 64초과 , 96 이하) + (x값이 512 초과 560 이하) + (y-z = 2(x-y)) 를 만족하는 값을 x,y,z 값으로 넣어주야 합니다. 따라서 저는 아래와 같이 구성하였습니다.

```python
def solve():
    prog = Program("./problem2.bin")
    print(prog.read_line()) # Read the initial message of the program.
    prog.send_line("94 378 520")
    print(prog.read_line())
```

```
cse20210354@cspro2:~/컴시개/Lab2/2-2$ ./problem2.bin
Provide your input:
94 378 520
You passed the challenge!
```

이렇게 x,y,z 를 구성해서 모든 조건을 만족했습니다.

Problem 2.3

```
0x0000000000401136 <+0>:     push   %rbp
0x0000000000401137 <+1>:     push   %rbx
0x0000000000401138 <+2>:     sub    $0x18,%rsp
0x000000000040113c <+6>:     mov    $0xa,%ebp
0x0000000000401141 <+11>:    mov    $0x0,%ebx
0x0000000000401146 <+16>:    jmp    0x40114e <main+24>
```

%rbp 와 %rbx 를 스택에 넣은 뒤 스택 자리를 확보해주고나서 ebp 레지스터에 0xa(10) 과 ebx 에 0값을 설정합니다. 그 이후에 바로 main<24> 로 jmp 합니다.

ebx 가 2와 비교해서 더 큰 경우에는 main 98 로 jmp 하는데 main 98 에서는 별다른 연산 없이 함수를 마무리 짓기 때문에 ebx >2 인 경우는 함수 자체를 종료하는 부분입니다. – (아마도 for 문에서 I <3 일때까지 루프를 도는데 ebx 가 I 에 해당하는 변수로 보는게 꽤나 합리적이라고 추론했습니다)

```
0x0000000000401148 <+18>:    add    $0x1,%ebp
0x000000000040114b <+21>:    add    $0x1,%ebx
0x000000000040114e <+24>:    cmp    $0x2,%ebx
0x0000000000401151 <+27>:    jg     0x401198 <main+98>
0x0000000000401153 <+29>:    mov    $0x402004,%edi
0x0000000000401158 <+34>:    callq  0x401030 <puts@plt>
0x000000000040115d <+39>:    lea    0xc(%rsp),%rsi
0x0000000000401162 <+44>:    mov    $0x402018,%edi
0x0000000000401167 <+49>:    mov    $0x0,%eax
0x000000000040116c <+54>:    callq  0x401040 <__isoc99_scanf@plt>
```

다시 돌아가서 , Ebx<2 인 경우를 살펴보면 put 함수를 호출합니다 ("provide your input ") 그리고 나서 lea 0xc(%rsp) , %rsi 를 통해서 스택 주소를 넘겨주고 (rsi 는 추후 scanf 함수에서 변수로 사용되기 때문) eax 를 0으로 설정 , edi ("%d") 를 설정해준 뒤 scanf 함수를 call 합니다. 그리고 다시 복귀하면 rsi 값에는 입력한 어떤 값이 들어 있을 것 입니다.

```
0x0000000000401171 <+59>:    mov    0xc(%rsp),%eax
0x0000000000401175 <+63>:    cmp    $0x7,%eax
0x0000000000401178 <+66>:    ja     0x40114b <main+21>
0x000000000040117a <+68>:    mov    %eax,%eax
0x000000000040117c <+70>:    jmpq   *0x402060(,%rax,8)
0x0000000000401183 <+77>:    sub    $0x1,%ebp
0x0000000000401186 <+80>:    jmp    0x40114b <main+21>
0x0000000000401188 <+82>:    add    %ebp,%ebp
0x000000000040118a <+84>:    jmp    0x40114b <main+21>
-Type <RET> for more, q to quit, c to continue without paging--
0x000000000040118c <+86>:    imul   %ebp,%ebp
0x000000000040118f <+89>:    jmp    0x40114b <main+21>
0x0000000000401191 <+91>:    mov    $0xa,%ebp
0x0000000000401196 <+96>:    jmp    0x40114b <main+21>
0x0000000000401198 <+98>:    cmp    $0x191,%ebp
0x000000000040119e <+104>:   je     0x4011b6 <main+128>
0x00000000004011a0 <+106>:   mov    $0x402038,%edi
0x00000000004011a5 <+111>:   callq  0x401030 <puts@plt>
0x00000000004011aa <+116>:   mov    $0x0,%eax
0x00000000004011af <+121>:   add    $0x18,%rsp
0x00000000004011b3 <+125>:   pop    %rbx
0x00000000004011b4 <+126>:   pop    %rbp
0x00000000004011b5 <+127>:   retq
0x00000000004011b6 <+128>:   mov    $0x40201b,%edi
0x00000000004011bb <+133>:   callq  0x401030 <puts@plt>
0x00000000004011c0 <+138>:   jmp    0x4011aa <main+116>
```

입력받은 값을 eax 레지스터에 옮긴 뒤 해당 값과 0x7 과 비교해서 7보다 큰 경우에는 <main21> 로 jump 합니다. 여기서 main 21 은 ebx 를 +1 하는데, 이는 for 문에서 i++ 하는 것과 같다고 보면 될 것 같습니다. 그리고 이전과 똑같이 for 문을 돌아줍니다.

다시 돌아가서 입력값이 7 이하인 경우에는 jmpq *0x402060(,%rax,8) 를 진행하게 됩니다. 이는 저희가 배웠던 Jump Table 이라고 생각했습니다. 따라서 저는 그 뒤에 어셈블리 코드를 해석하기 이전에 먼저 jump table 을 찍어보았습니다. (또한 교수님께서 말씀해주셨듯이 과제 4개중 하나에는 jump table 이 존재하신다고 하셨는데 이 이야기를 듣고 더 염두해두고 풀었습니다.)

```
(gdb) x/16gx 0x402060
0x402060:       0x0000000000401148      0x000000000040114b
0x402070:       0x0000000000401183      0x0000000000401188
0x402080:       0x000000000040114b      0x0000000000401191
0x402090:       0x000000000040114b      0x000000000040118c
```

다음과 같이 jump table 이 나옵니다. 이제 jump table 을 해석해보도록 하겠습니다.

Rax = 0 인경우 401148 : add %0x1 , %ebp → 입력값 ++ 후 다시 for 루프 돈다.

Rax = 1 인경우 40114b : i++ (즉 다시 for 루프를 도는 것)

Rax = 2 인경우 401183 : 입력값 -=1 후 다시 for 루프 돈다.

Rax = 3 인경우 401188 : 입력값 *=2 후 다시 for 루프 돈다.

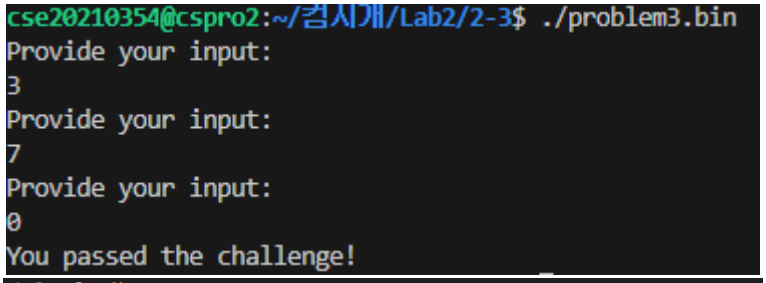
Rax = 4 인경우 40114b : rax 가 1인경우와 동일.

Rax = 5 인경우 401191 : ebp 를 0xa(10)으로 설정

Rax = 6 인 경우 40114b : rax가 1인 경우와 동일

Rax =7 인경우 40118C : 입력값의 제곱 반환

이제 FOR 루프를 다 돌고 난 뒤를 생각해보면, 앞에서 설명했듯이 EBX > 2 인 상황이 될 것입니다. 따라서 <main 98> 로 jump 합니다. 최종적으로 ebp 가 0x191(401) 과 같으면 0x4011b6 으로 jump 하게 되고 아닌 경우에는 <106> 으로 간 뒤 서로 다른 메시지로 puts 함수를 호출한 뒤 스택을 복구하고 ret 합니다. 결과적으로 저는 초기값이 10인 %ebp 를 for 루프를 돌면서 401 로 만들어야 통과할 수 있다고 판단했습니다. 따라서 10 -> 20 -> 400 -> 401 이렇게 곱하기 2 , 제곱 , +1 연산을 한다면 3번만에 10 을 401 로 만들 수 있다고 판단했고 이에 따라서 적절한 입력을 해주었습니다.




```python
def solve():
    prog = Program("./problem3.bin")
    print(prog.read_line()) # Read the initial message of the program.
    prog.send_line("3")
    print(prog.read_line(), end="")
    prog.send_line("7")
    print(prog.read_line(), end="")
    prog.send_line("0")
    print(prog.read_line())
```

그래서 다음과 같이 함수들을 구성하였고. Passed the challenge 결과값 까지 확인했습니다 !.

Problem 2.4

이전문제들과 유사하게 시작합니다. <+16>: lea 0x20(%rsp),%rsi 를 한뒤 scanf 함수를 호출해서 스택 값에 buf 에 해당하는 값(주소)를 받아옵니다. 그뒤에 <+36>: lea 0x20(%rsp),%rdi 이후에 strlen 함수를 호출에서 c 코드에 있는 n=strlen(buf) 를 실행합니다.

그 뒤에 eax 값을 ebx 로 옮겨두고 eax 값(문자열 길이)와 $0xb(11) 값을 비교해서 둘이 같은 경우에는 <main 88> 로 jump 합니다. 이 경우에는 ebp 를 1로 설정하고 , 아닌 경우에는 ebp 가 0을 가지게 됩니다. 그 다음 edx값을 0x1a(26)으로 설정하고 esi 값을 0으로 바꿔줍니다. 그 뒤에 %rsp값을 %rdi 값에 mov 해 주고 memset 함수를 call 합니다. 이름으로 미루어보아 c 언어의 memset 함수일 것 같아서 함수를 초기화 시켜주려고 한다고 유추했습니다. (아마 함수내에서 인자로 rsi 와 rdi + edi를 사용할 것인데 저희가 이전에 초기화된 인자를 memset 이 사용했을 것 으로 생각했습니다.)

```
0x0000000000401189 <+51>:    je     0x4011ae <main+88>
0x000000000040118b <+53>:    mov    $0x0,%ebp
0x0000000000401190 <+58>:    mov    $0x1a,%edx
0x0000000000401195 <+63>:    mov    $0x0,%esi
0x000000000040119a <+68>:    mov    %rsp,%rdi
0x000000000040119d <+71>:    callq  0x401050 <memset@plt>
0x00000000004011a2 <+76>:    mov    $0x0,%esi
0x00000000004011a7 <+81>:    mov    $0x0,%ecx
0x00000000004011ac <+86>:    jmp    0x4011b8 <main+98>
0x00000000004011ae <+88>:    mov    $0x1,%ebp
0x00000000004011b3 <+93>:    jmp    0x401190 <main+58>
0x00000000004011b5 <+95>:    add    $0x1,%ecx
0x00000000004011b8 <+98>:    cmp    %ebx,%ecx
0x00000000004011ba <+100>:   jge    0x4011fe <main+168>
```

그 뒤에 esi 를 0으로 설정하고 ecx 를 0으로 설정합니다. 그다음 <86>에서 <main 98> 로 jump 합니다. ebx(문자열길이) 와 ecx(정확히 아직 모름)을 비교해서 ecx 가 더 크거나 같은 경우에는 <main 168> 로 jump 합니다. 이 경우는 뒤에서 더 자세히 설명하겠습니다. 대략 루프를 탈출하고 ret 합니다. (아마 탈출하는 조건)

```
0x00000000004011bc <+102>:   movslq %ecx,%rax
0x00000000004011bf <+105>:   movzbl 0x20(%rsp,%rax,1),%eax
```

더 작은 경우에는 rax 에 ecx 값을 mov 한 뒤에 movzbl0x20(%rsp,%rax,1),%eax 연산을 통해서 eax 값을 변경해주는데 0x20+rsp+rax 값은 결국 저희가 scanf 로 입력받은 buf 에서 rax 만큼 떨어진 문자열을 가르키고 있을 것 입니다. (문제 1번과 유사) . 그 값을 eax 에 저장합니다.

```
0x00000000004011c4 <+110>:   lea    -0x61(%rax),%edx
0x00000000004011c7 <+113>:   cmp    $0x19,%dl
0x00000000004011ca <+116>:   jbe    0x4011d1 <main+123>
0x00000000004011cc <+118>:   mov    $0x0,%ebp
0x00000000004011d1 <+123>:   mov    %ebx,%edx
0x00000000004011d3 <+125>:   sub    %ecx,%edx
0x00000000004011d5 <+127>:   sub    $0x1,%edx
0x00000000004011d8 <+130>:   movslq %edx,%rdx
0x00000000004011db <+133>:   cmp    %al,0x20(%rsp,%rdx,1)
0x00000000004011df <+137>:   je     0x4011e6 <main+144>
```

이후에 (-0x61+rax) 을 edx 에 옮겨줍니다. 이게 위 문제에서 해석하기 어려웠는데. -0x61 이 정확히 어떤 의미인지 파악하기 힘들었습니다. 그래서 아스키코드 표를 참고했고 0x61 이 a 라는 문자열이라는 것을 알고 난 뒤에는 c언어에서 자주 쓰이는 문법인 (char형 – 'a') 과 같은 형태임을 떠올렸습니다.

그래서 rax-'a' 값을 edx 에 저장해줍니다. 그러면 a 는 0 … z는 25 인 int 형이 저장되게 됩니다. 그리고 0x19 값과 dl 을 비교합니다. 만약 dl 값이 25보다 작으면 (즉, 여기서는 소문자 영어라면) <main 123> 으로 jump 합니다. (만약 아닌 경우에는 ebp 를 0으로 설정) 그 다음에 ebx(문자열 길이) 를 edx 에 옮겨줍니다. 그다음에 sub %ecx , %edx 연산으로 문자열 길이 – 현재 인덱스(edx) 를 빼주고 추가로 1을 더 빼줍니다. 그리고 그 값을 rdx 에 옮겨줍니다. 그 후에 al 과 0x20(%rsp,%rdx.1) 값을 비교합니다. 조금 더 이해하기 쉽게 하기 위해서 수식을 작성해보면 ,

Edx = n (= strlen(buf)) , ecx = i 라고 가정하면

Edx = n – I – 1 ; 이 됩니다.

후에 cmp 를 다시 c코드와 유사하게 적어보면

Buf[edx] (= buf[n-i-1]) == dl (buf[i]) 이므로 저희가 작성해야할 문자열은 대칭이여야 한다는 것을 추론 할 수 있습니다.

일단 I 와 n-i-1 의 인덱스가 같은 값이라면 이라면 <main 144> 로 jmp 합니다.

```
0x00000000004011e6 <+144>:   movsbl %al,%eax
0x00000000004011e9 <+147>:   sub    $0x61,%eax
0x00000000004011ec <+150>:   movslq %eax,%rdx
0x00000000004011ef <+153>:   cmpb   $0x0,(%rsp,%rdx,1)
0x00000000004011f3 <+157>:   jne    0x4011b5 <main+95>
0x00000000004011f5 <+159>:   movb   $0x1,(%rsp,%rdx,1)
0x00000000004011f9 <+163>:   add    $0x1,%esi
```

위와 유사하게 eax-'a' 를 연산해 해당 값을 rdx 에 저장합니다.

저장한 뒤에 0 과 rsp+rdx 값을 비교합니다. 이는 아까 위에서 스택에 있는 값들을 초기화 해둔 값에서 rdx 값에 해당하는 인덱스에 값이 0이 아닐때는 (즉, 값을 바꿨을 경우에는) 에는 <main 95> 로 jump 하는 것 입니다. 만약 바뀐적이 없다면 해당 인덱스 값을 1 로 바꿔줍니다.

그리고 esi 값을 +1 올려줍니다. 후에 다시 <main 95> 로 돌아가서 루프를 다시 돕니다.

```
0x00000000004011fe <+168>:   cmp    $0x5,%esi
0x0000000000401201 <+171>:   jne    0x401207 <main+177>
0x0000000000401203 <+173>:   test   %ebp,%ebp
0x0000000000401205 <+175>:   jne    0x40121d <main+199>
0x0000000000401207 <+177>:   mov    $0x402038,%edi
0x000000000040120c <+182>:   callq  0x401030 <puts@plt>
0x0000000000401211 <+187>:   mov    $0x0,%eax
0x0000000000401216 <+192>:   add    $0x48,%rsp
0x000000000040121a <+196>:   pop    %rbx
0x000000000040121b <+197>:   pop    %rbp
0x000000000040121c <+198>:   retq
```

마지막 부분에 esi 가 아니라면 177 로 jmp 하고 5인 경우에는 (더하는 경우가 문자열 대칭인 경우 + 총 문자열이 11글자니까 대칭이면서 11 글자인 경우 + 사용한 문자열인 경우 해당 인덱스 값이 1 이 되어서 다른 분기로 이동하기 때문에 서로다른 문자열 종류를 5개 쓴경우) 를 만족합니다. 아닌 경우에는 아마 fail 이 뜰 것입니다. (직접 input 값으로 넣어보았습니다.) 너무 길어져서 ret 하는 부분은 이전과 유사해서 생략했습니다.

다시 정리하면

1. 입력 문자열이 소문자

2. 11글자

3. 문자열이 대칭

4. 서로다른 문자 5개 쓴경우

이 3가지 조건을 모두 만족하는 값을 입력으로 넣었습니다.

```
cse20210354@cspro2:~/컴시개/Lab2/2-4$ ./problem4.bin
Provide your input:
abcdeeedcba
You passed the challenge!
```

<최종결과>

```
cse20210354@cspro2:~/컴시개/Lab2$ ./check.py
[*] 2-1 : 0
[*] 2-2 : 0
[*] 2-3 : 0
[*] 2-4 : 0
```