Lab2 report

20190808 방지혁

Lab 2-1)

우선 %rsp를 40바이트만큼 빼면서 스택을 할당한다. 이는 sub $0x28, %rsp 명령어이다. 32바이트의 char buf[32] 배열과 추가적인 임시 공간을 위해 할당하는 것이다

다음 두 줄은 puts함수를 위한 줄으로, 인자로 들어가는 %edi를 x/s 0x402004로 분석해보면 "Provide your input:"이 들어가 있고, 이를 인자로 넣어 출력하는 부분임을 알 수 있다.

그 후 다음 4줄은 scanf 함수를 위한 줄인데 두번째 인자인 %rsi에 %rsp를 넣어, buf의 시작 주소를 인자로 전달하고 두번째 인자인 %edi는 x/s 0x402018을 통해 확인하면, "%31s"가 들어가 있음을 확인할 수 있다. 즉, 입력을 통해 31바이트의 문자열을 읽는 것이고 뒤에는 NULL 문자가 추가될 것이다.

그 다음부터는 반복문이다. 카운터로 사용될 %eax에 0을 넣어 초기 조건을 설정한다. 이를 다시 %rdx로 옮긴다. 루프가 다시 반복될 때에는 해당 주소로 점프해서 add $0x1, %eax를 통해 1이 늘어난 값이 들어갈 것이다.

다음 6줄은 0x404038에 들어가 있는 정답인 문자열과 1바이트씩 비교하는 부분이다. movzbl 0x404038(%rdx), %edx를 분석해보자면, 0x404038 + i에서 정답 문자열의 i번째 문자를 읽어 %edx에 저장한다. 또한, test %dl, %dl 줄을 분석해보자면 읽은 문자가 널 종료 문자(₩0)인지 확인한다. 만약 해당 문자가 NULL문자라면 0x401176으로 점프하여 루프를 탈출한다.

NULL문자가 아닐 경우 movslq %eax, %rcx줄에서 i를 %rcx에 옮기고, cmp %dl, (%rsp, %rcx,1)에서 정답인 문자열의 문자가 저장되어 있는 %dl과 buf[i]를 의미하는 %rsp + i와 비교한다. 만약 다르다면 0x401193으로 jump 하는데 이는 0을 %eax에 넣고 실패 문자열을 출력한다. 이는 x/s 0x402038을 통해 확인하면 알 수 있다. 이 때는 다시 0x401189로 jump하여 0을 return하고 add $0x28, %rsp를 통해 스택을 복구하고 함수를 종료한다.

만약 같을 시 루프 카운터에 add $0x1, %eax를 통해 1을 추가하며 다음 바이트를 계속 비교해준다.

만약 바이트가 ₩0이라면 0x401176으로 jump하며 루프를 탈출한다. %eax에는 1이 들어가 있기에 0x40119a로 jump하지 않고 puts함수 부분으로 간다. Puts 함수 인자인 0x40201d를 확인해보면 "you passed the challenge!"라고 들어가 있기에 성공했다는 것을 알 수 있다. Return 값인 %eax에 0을 넣어주고 스택을 정리하며 함수를 종료한다.

즉 정답 문자열인 0x404038을 확인해보면 "CSE3030@Sogang"을 입력해줘야 함을 알 수 있다.

Lab 2-2)

0x401136 <+0>: sub $0x18, %rsp 이 줄은 지역 변수를 위해 stack pointer에서 24 byte를 빼는 부분이다. 이후 0x40113a <+4>: mov $0x402004, %edi, 0x40113f <+9>: call 0x401030 <puts@plt>이 두 줄이 있다. x/s 0x402004를 통해 확인해보면 "Provide your input:"이 들어가 있음을 확인할 수 있고, 이를 인자로 넣어 출력하는 부분임을 알 수 있다. 이후 0x40115d <+39>: call 0x401040 <__isoc99_scanf@plt의 인자로 %rdi, %rsi, %rdx, %rcx가 들어가야 하는데, 이전에 lea를 통해 %rsi에 0xc(%rsp), %rdx에 0x8(%rsp), %rcx에 0x4(%rsp)를 넣어준다. 이는 c 함수로 본다면 &x, &y &z와 같다. %rdi에는 $0x402018를 넣는데, 이를 x/s 0x402018을 통해 확인해보면 %d %d %d를 인자로 넘겨주는 것을 확인할 수 있다.

그 후 본격적인 조건 및 분기로 들어간다. 0x401162 <+44>: mov 0xc(%rsp), %edx를 통해 x 값을 %edx에 넣어준다. 이를 0x40(64)와 비교하는데 (cmp $0x40, %edx 줄), x <= 64라면, flag 변수인 %ecx를 0으로 설정하고 아닌 경우 1로 설정한다 (mov $0x1, %ecx). 또한 0x60(96)과 비교하는데 만약 x <= 96이라면 %ecx를 1로 설정하고 아닌 경우 0으로 설정한다. 즉 flag 변수인 %ecx는 64 < x <=96인 경우 1이고 아닌 경우 0이다.

그 후 mov 0x4(%rsp), %eax절에서 z 값을 %eax에 옮기고 512와 비교하여 512초과인지 확인한다. 만약 z > 512이면 %ecx를 0으로 설정한다. 이후 560과 비교하는데 z > 560이면 마찬가지로 0으로 설정한다. 즉 flag가 1이 되기 위해서는 64 < x <=96이며 512 < z <= 560이어야 하는 것이다.

%esi에 y값을 옮기고, 이를 또 %edi에 옮긴다. %edi에서 x를 빼 y-x를 계산하고, %eax에는 z가 원래 들어가 있는데 yf를 빼고 이를 2배하여 2 * (z – y)를 계산하여 둘의 값을 비교한다. 만약 다르다면 0x4011aa로 jump하는데 mov $0x402040, %edi / call 0x401030 <puts@plt> / mov $0x0, %eax하는 부분이다. %edi 값을 보면 실패했다는 것을 알 수 있다. 다시, 같은 경우 %ecx값을 test하는데 해당 값이 1인 경우 0x4011c5로 jump해 성공 메시지를 출력한다. 다시 0x4011b4로 돌아와 %eax(return 값)에 0을 넣어주고 스택을 복구하고 종료한다.

정리하자면 64 < x <= 96, 512 < z <= 560, (y - x) == 2 * (z - y) 세 조건을 만족해야 하며, 이는 66 364 513이다.

Lab 2-3)

Assembly Code	Analysis
0x401136 <+0>: push %rbp	이전 caller의 %rbp를 스택에 저장하여 값을 보존한다.
0x401137 <+1>: push %rbx	이전 caller의 %rbx를 스택에 저장하여 값을 보존한다.
0x401138 <+2>: sub $0x18, %rsp	Local variable 할당을 위해 %rsp를 24바이트만큼 빼준다.
0x40113c <+6>: mov $0xa, %ebp	%ebp에 10(0xa)을 저장하는데, 이는 예제 코드에 없는 새로운 변수 temp에 해당한다.
0x401141 <+11>: mov $0x0, %ebx	%ebx에 0을 저장하는데, 이는 i에 해당한다.
0x401146 <+16>: jmp 0x40114e <main+24>	loop를 돌 때 i의 조건을 확인하기 위해, 해당 주소로 jump한다,
0x401148 <+18>: add $0x1, %ebp	**Jump table에서 x가 0이면** 이 주소로 jump 하여 temp 변수를 1 증가시킨다.
0x40114b <+21>: add $0x1, %ebx	**Jump table에서 x가 1, 4 또는 6이면** case문에서 break 하고 for문의 반복을 위해 i에 1을 더한다.
0x40114e <+24>: cmp $0x2, %ebx	c코드의 for loop에서 i < 3 임을 확인하는 부분과 같다.
0x401151 <+27>: jg 0x401198 <main+98>	i > 2인 경우 i는 3이기에 loop 반복문에서 탈출하여 해당 주소로 jump한다.
0x401153 <+29>: mov $0x402004, %edi 0x401158 <+34>: call 0x401030 <puts@plt>	"Provide your input:"이 0x402004에 저장되어 있고, 이를 puts의 인자로 넘겨서 출력한다.
0x40115d <+39>: lea 0xc(%rsp), %rsi 0x401162 <+44>: mov $0x402018, %edi 0x401167 <+49>: mov $0x0, %eax 0x40116c <+54>: call 0x401040 <생략...scanf...생략>	scanf의 첫번째 인자로 %d, 두번째 인자로, 0xc + %rsp 주소(&x)를 넘겨준다 입력이 된다면 해당 주소에 저장될 것이다.
0x401171 <+59>: mov 0xc(%rsp), %eax 0x401175 <+63>: cmp $0x7, %eax	0xc + %rsp 주소의 메모리에 저장되어 있는 값을 %eax에 넘기고, 이를 7과 비교한다.
0x401178 <+66>: ja 0x40114b <main+21>	x가 7보다 큰 경우인 default의 경우로 0x40114b로 jump 한다. case문의 default에 해당한다.
0x40117a <+68>: mov %eax, %eax	x가 7보다 작거나 같은 경우로 %eax을 자기자신에 복사한다.
0x40117c <+70>: jmp *0x402060(,%rax,8)	x의 값에 따라 해당 jump table에 있는 주소로 jump한다. Jump table은 아래에 첨부.
0x401183 <+77>: sub $0x1, %ebp 0x401186 <+80>: jmp 0x40114b <main+21>	**x가 2인 case로** temp 변수를 1 감소시키고, case문에서 break하여 for문의 처음으로 돌아간다.
0x401188 <+82>: add %ebp, %ebp 0x40118a <+84>: jmp 0x40114b <main+21>	**x가 3인 case로** temp 변수를 2배 늘리고, case문에서 break하여 for문의 처음으로 돌아간다.
0x40118c <+86>: imul %ebp, %ebp 0x40118f <+89>: jmp 0x40114b <main+21>	**x가 7인 case로** temp 변수를 제곱하고, case문에서 break하여 for문의 처음으로 돌아간다.
0x401191 <+91>: mov $0xa, %ebp 0x401196 <+96>: jmp 0x40114b <main+21>	**x가 5인 case로** temp 변수에 10을 넣고, case문에서 break하여 for문의 처음으로 돌아간다.
0x401198 <+98>: cmp $0x191, %ebp 0x40119e <+104>: je 0x4011b6 <main+128>	for loop을 탈출하고 난 후, temp 변수(%ebp) 0x191(10진수로 401)과 값을 비교한다. 만약 값이 401과 같다면 0x4011b6로 jump한다.
0x4011a0 <+106>: mov $0x402038, %edi 0x4011a5 <+111>: call 0x401030 <puts@plt>	x/s를 통해 내용 확인 시 "No, that is not the input I want!"가 저장되어 있고, 이를 %edi에 인자로 전달하여 puts함수에서 출력한다. 즉 실패했다는 뜻이다.
0x4011aa <+116>: mov $0x0, %eax 0x4011af <+121>: add $0x18, %rsp 0x4011b3 <+125>: pop %rbx 0x4011b4 <+126>: pop %rbp 0x4011b5 <+127>: ret	문제의 조건에 성공을 했던 실패를 했던 반환값인 %eax에 0을 넣어주고, %rsp(스택 포인터)를 24바이트 증가시켜 지역 변수 공간을 해제한다. 또한, %rbx, %rbp를 복원하며 함수를 종료한다.
0x4011b6 <+128>: mov $0x40201b, %edi 0x4011bb <+133>: call 0x401030 <puts@plt>	x/s 0x40201b를 통해 내용을 확인해보면 "You passed the challenge!"가 저장되어 있다. 이를 %edi에 인자로 전달하여 puts함수에서 이를 출력한다. 이는 즉 문제의 조건에 temp의 값이 401로 부합한다는 뜻이다.
0x4011c0 <+138>: jmp 0x4011aa <main+116>	0x4011aa로 jump한다.

x/7xg 0x402060를 통해 확인 가능			
0x401148	**%rax == 0**	**0x40114b**	**%rax == 4**
0x40114b	**%rax == 1**	**0x401191**	**%rax == 5**
0x401183	**%rax == 2**	**0x40114b**	**%rax == 6**
0x401188	**%rax == 3**	**0x40118c**	**%rax == 7**

temp를 401로 만들기 위해서는 초기 값이 10인 temp를 20으로 만들어야 한다. 그래서 3을 입력한다. 이를 제곱하여 400으로 만들어야 하기 때문에 7을 입력하고, 1을 더해 401로 만들어줘야 하기 때문에 0을 입력한다.

Lab 2-4)

초기 스택 프레임 설정, puts, scanf: 첫 두 줄 push %rbp과 push %rbx에서 알 수 있듯 우선 이전 caller의 %rbp와 %rbp를 스택에 저장하여 값을 보존한다. 그 다음 줄 sub $0x48, %rsp에서는 Local variable 할당을 위해 %rsp를 72바이트만큼 빼준다. 그다음 $0x402004를 %edi에 넣어 puts함수에 인자로 넘겨주어 "Provide your input:"을 출력한다. 이후 다음 4줄에서는 scanf함수를 호출하는데, 이 인자로는 %rsi에 0x20(%rsp)를, %edi에 $0x402018를 넘겨주고 %eax는 0으로 설정하여 부동소수점 인자가 없음을 나타낸다. 이를 마찬가지로 확인해보면 %edi에는 %31s가 들어가 있으며 0x20(%rsp)는 c 코드에서 buf의 주소임을 알 수 있다.

Strlen: 이후 lea 0x20(%rsp), %rdi 줄에서 보면 buf의 주소를 %rdi에 넣어, strlen 함수의 인자로 넘겨 호출한다. strlen 함수는 해당 문자열의 길이를 계산하여 %eax에 저장하는데 이 값을 %ebx에 저장한다. 이는 c 코드에서 n과 같다.

Memset: cmp $0xb, %eax를 통해 문자열의 크기와 11(0xb)을 비교한다. 만약 같다면, je 0x4011ae 줄에서 볼 수 있듯 해당 주소로 jump한다. 우선 문자열의 크기가 11인경우에 대해 서술해보겠다. 0x4011ae로 jmp하여 %ebp를 1로 설정하고, 0x401190으로 jump한다. 해당 부분부터 4줄은 memset 함수에 넘길 인자를 설정하는 부분인데, 첫번째 인자인 %rdi로 %rsp, 두 번째 인자인 %rsi로 0을 넣는다. 이는 해당 배열을 0으로 초기화한다는 뜻이다. 그리고 %edx에 0x1a(26 byte)를 넘겨 배열의 크기를 넘겨준다. 11이 아닌 경우, %ebp는 0으로 설정하고 memset 부분은 똑같이 이루어진다. 이후 %rsi, %rcx에 0을 넣고 0x4011b8로 jump한다.

For loop 탈출 조건 체크: 해당 줄에서 %rcx - %rbx 값이 0보다 크거나 같은 경우, 즉 %rcx가 %rbx보다 크거나 같은 경우 0x4011fe로 jump한다. 이는 루프 카운터인 %rcx가 문자열의 길이 n인 %rbx보다 크거나 같은 경우 루프를 탈출하는 부분이라고 볼 수 있다.

For loop 내 알파벳 확인: movslq %ecx, %rax / movzbl 0x20(%rsp, %rax, 1), %eax / lea -0x61(%rax), %edx 해당 3줄을 보면 %ecx를 %rax에 넘기는데 이는 루프 카운터 값이다. 다시 %eax에 0x20 + %rsp + %rax주소에 있는 값을 넘겨주는데 이는 buf[%ecx]에 있는 문자를 넘겨주는 것을 의미한다. 해당 값에서 0x61을 빼는데 이는 ASCII 값에서 'a'를 빼 인덱스를 계산하는 것이라고 볼 수 있다. 만약 결과가 25(0x19)이하가 아니라면 flag인 %ebp를 0으로 설정한다. 즉 문자가 소문자 알파벳인 a ~ z인지 확인하고 그렇지 않으면 유효하지 않기에 flag를 0으로 설정하는 것이다.

For loop 내 회문(palindrome) 검사: 알파벳 조건 검사를 통과하면 0x4011d1으로 jump하는데 %ebx에 들어있는 n 즉, 배열의 크기를 %edx에 넘겨주고 %ecx(루프 카운터)와 1을 빼, buf[n – 1 - 루프카운터]와 buf[루프카운터]를 비교해준다. 이는 어셈블리 코드 상에서 movslq %edx, %rdx, cmp %al, 0x20(%rsp, %rdx, 1)부분과 같다. 만약 다를 시 마찬가지로 flag를 0으로 설정해주고 같으면 0x4011e6으로 jump한다.

For loop 내 고유 문자 검사: %al을 %eax로 옮기고 0x61을 빼는데 이는 이전에 있던 'a'를 빼 index를 확인하는 부분과 같다. 해당 index를 %rdx에 옮겨, 이전에 우리가 memset함수로 만들었던 알파벳 카운터 배열의 index에서 해당 index가 0과 비교하여 0인지 확인한다. (movsbl %al, %eax / sub $0x61, %eax / movslq %eax, %rdx, cmpb $0x0, (%rsp, %rdx, 1)) 만약 0이면 해당 인덱스의 값을 1로 설정해주고 %esi(고유문자 개수 카운터)를 1 증가시킨다.

For loop 이후: flag 변수 %esi(고유문자 개수 카운터)가 5이며, %ebp(다른 검사 flag)가 0이 아닌지 확인한다. 이는 우리가 고유문자가 정확히 5개이며, palindrome이며, 문자열 길이가 11이어야한다는 조건이다.

만약 부합하지 않을 시 0x401207로 jump하고 부합할 시 0x40121d로 jump하는데, mov $0x40201d, %edi 절에서 인자로 들어가는 %edi를 x/s 0x40121d를 확인해보면 You passed the challenge!가 들어가 있기 때문에 성공했음을 확인할 수 있다. 이후 0x401211로 jump 해서 return 값을 0으로 설정하고 스택을 복원한 후 함수를 종료한다.