2018년 2학기

컴퓨터 SW시스템 개론

CSED211

Lab Report #3

담당교수 : 김종

학번 : 20160074

학과 : 화학공학과

이름 : 고진민

POVIS ID : eric9709

--------------------------------------------------------------------------------------------------------------

<명예서약>

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

--------------------------------------------------------------------------------------------------------------

1. **지난주에 한 내용**

지난 시간에는 어셈블리어에 대해서 본격적으로 배웠다. 어셈블리어는 소스 코드를 기계어로 컴파일한 결과를 사람이 볼 수 있도록 변환한 것이다. 레지스터는 빠른 저장과 접근이 가능한 작은 용량의 저장소이다. 다양한 레지스터가 있으며 이 중 rsp는 stack pointer를 저장하는 레지스터이다. rdi, rsi, rdx, rcx는 순서대로 1~4번째 argument를 저장하는 레지스터이고 rax는 함수의 return value를 저장하고, rbx는 calle saved이다. rax를 기준으로, rax 안에는 eax, ax, al이라는 레지스터가 있는데 이들은 저장 용량이 다르다. 먼저 rax는 8byte, eax는 4byte, ax는 2byte, al은 1byte이다.

Assembly operation code는 줄 당 operation의 단위이다. 이항연산의 경우, OP SRC, DST 와 같이 표현하고(op는 연산자) 이는 DST = OP(SRC, DST) 와 동치이다. 단항연산의 경우 OP DST 와 같으며 이는 DST = OP(DST) 와 동치이다. 피연산자는 상수, 레지스터, 메모리 등이 있다. 연산자에는 mov, add, sub, imul, cltq, lea 등이 있다. 순서대로 복사, 덧셈, 뺼셈, 곱셈, Convert long to quad, load effective address이다.

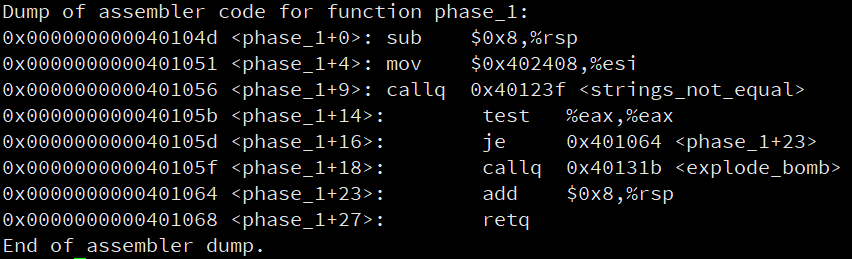
연산자 이외에도 비교와 점프가 있다. 비교는 cmp로 이루어지며 비교 결과에 따라서 CF, ZF, SF, OF 4가지 flag에 정보가 저장된다. 순서대로 carry flag, zero flag, sign flag, overflow flag이다. 이들의 정보를 토대로 점프 여부가 결정된다. 가령 jne의 경우, not equal, not zero인 경우에 점프가 일어나므로 ~ZF값에 따라 점프 여부를 결정한다.

Stack은 메모리에 있고, 변수가 stack에 저장될 수 있다. 소스코드가 일반적으로 변환된 후 stack에 저장된다. 또한 아래로 자란다.

GDB는 GNU 기반 디버거이다. 리눅스 시스템 하에서 돌아가며 다양한 명령어를 사용 가능하다. 대표적으로 disas(disassemble), b(break), d(delete), r(run), c(continue), info(information), x(memory check) 등이 있다.

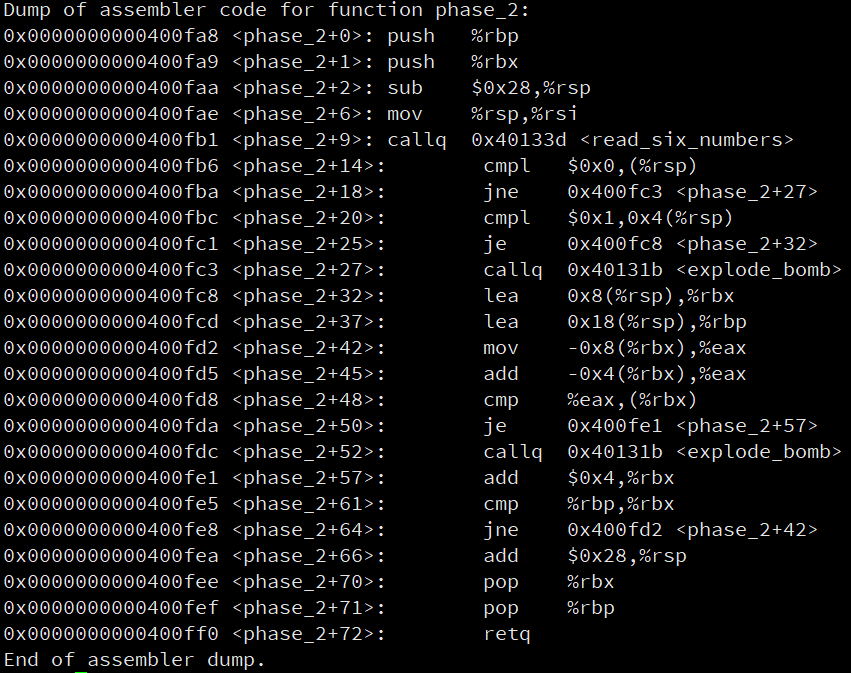
1. **Phase별 설명**

* **phase\_1**



Phase 1은 간단하다. 코드를 보면 +16에서 je가 충족되어야 explode bomb으로 진행하지 않으므로, strings\_not\_equal이 eax에 1을 반환해야 한다. 그러면 1st argument인 esi와 입력값이 같아야 하므로 0x42408에 있는 데이터를 뜯어서 그 값을 입력하면 된다. x/s를 이용하면, **He is evil and fits easily into most overhead storage bins.** 라는 결과를 얻을 수 있다. 이 값을 입력하면 phase 1이 해결된다.

* **phase\_2**



+9를 보면 입력값이 6개의 숫자임을 알 수 있다. 전체적인 구조를 보자. +14에서 0과 rsp에 저장된 주소에 있는 값을 비교한다. 이는 첫 입력 값이다. +18에서 비교 값이 다르면 폭탄이 터진다. 따라서 첫 입력 값은 0이어야 한다. +20에서 1과 rsp의 값에 4를 더한 위치의 값을 비교한다. 이는 두 번째 입력 값이다. +25에서 비교 값이 같아야 폭탄이 터지지 않는다. 따라서 두 번째 입력 값은 1이어야 한다.

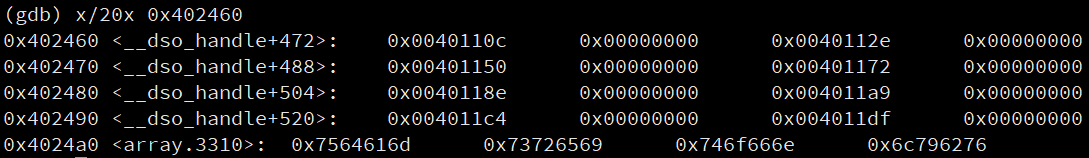
이후 rbx에 rsp에 8을 더한 값(주소)을 복사하고 rbp에 0x18을 더한 값을 복사한다. 이후 +42에서 eax에 (rbx-8)을 옮긴다. 이는 첫 번째 입력 값이다. +45에서는 eax에 (rbx-4)를 더한다. 이는 두 번째 입력 값이다. 이 값과 (rbx)를 비교하는데 rbx에 있는 주소는 세 번째 입력 값을 가리킨다. Rsp+8이기 때문. 같으면 폭탄이 터지지 않는다. 따라서 세 번째 입력 값은 0+1=1이다. 이후 rbx에 4를 더하고 +61에서 rbp와 rbx를 비교해서 다르면 +42로 돌아간다. 루프문인데, +61에서 두 값이 같으려면 루프를 4번 돈 후에 같아진다. 0x18에서 0x8을 빼고 4로 나누면 4회이기 때문. 루프에서 하는 과정을 잘 살펴보면 n번째 입력값이 n-1, n-2번째 입력 값과 같은지를 판별한다. 즉, 피보나치 수열이라고 생각할 수 있다. 입력 값은 순서대로 피보나치 수열이고 따라서 **0 1 1 2 3 5**가 입력 값이다.

* **phase\_3**

+19에서 esi에 옮겨지는 값을 뜯어보면 %d %c %d임을 알 수 있다. 즉, 순서대로 정수 문자 정수를 입력해야 한다. 이후 +34에서 eax를 확인하는데, 3 이상이어야 한다. 이는 입력한 개수가 맞는지 확인한 것이다. Sscanf는 입력한 값의 수를 반환하기 때문.

+44에서 7과 (rsp+0x14)를 비교하는데 뒤의 값은 우리가 처음에 입력한 값이다. 이후 ja가 있는데 ja는 jg의 unsigned 버전이므로 첫 번째 입력값이 더 크면 +304로 이동한다. 그런데 이곳으로 이동하면 폭탄이 터지므로 첫 번째 입력 값은 0 이상 7 이하여야 한다.

이후 eax에 첫 번째 입력 값을 옮기고 (0x402460+8\*eax)으로 점프한다. 즉, 이는 입력 값에 따라 수행되는 코드 위치가 다르다고 할 수 있다.



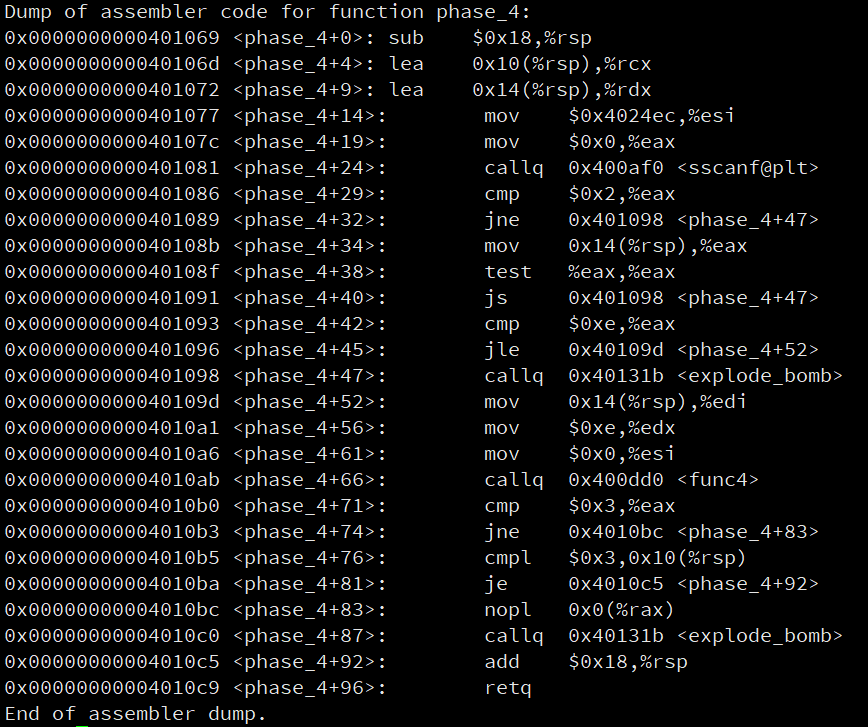
0x402460부터 20개의 값을 나열한 결과 0x4024a0부터는 의미가 없는 값임을 알 수 있다. 8개의 값을 일일이 계산하는 것보다는 brute-force 를 사용하거나 어떤 값이 맞는지 확인해서 위치를 역산하는 것도 가능하다. 어떤 구조인지 하나를 임의로 찾아 분석해보자. 만일 처음 입력한 값이 3인 경우, eax에 3이 있을 것이므로 0x402460+0x8 \* 3의 위치에 있는 0x401172(+168)로 점프할 것이다.

+168에서는 eax에 0x62를 옮긴 후 (rsp+0x10)와 0x3b의 값을 비교한다. 이 값은 세 번째로 입력한 문자이다. 0x3b는 십진수로 59이며, 따라서 세 번째로 입력한 값은 59여야 한다. 같아야 +320으로 점프하고 다르면 폭탄이 터진다.

+320에서는 (rsp+0xf)와 al을 비교한다. 이 값은 두 번째로 입력한 문자이다. al에는 앞서 +168에서 옮긴 0x62가 있다. ASCII 코드표에서 0x62에 해당하는 문자를 찾으면 b이다. 따라서 두 번째로 입력하는 문자는 b이다.

정리하면, 입력되는 값은 순서대로 **3 b 59** 이다.

* **phase\_4**

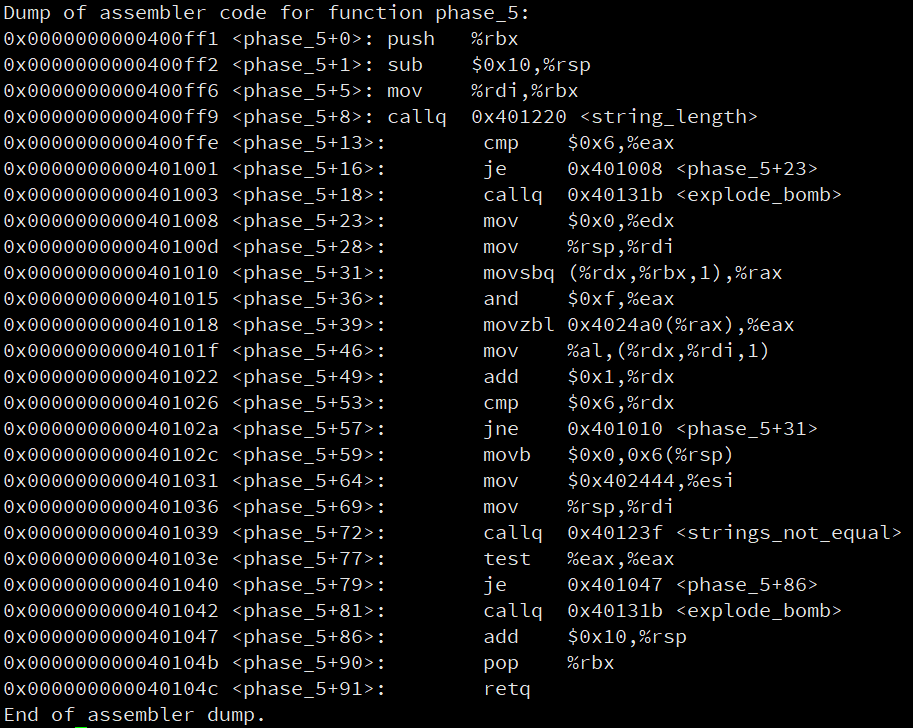


+14에서 0x4024ec를 뜯어보면 %d %d 라는 문자열이 나온다. 입력 값은 정수 2개이다. +29에서 입력된 값이 2개인지 확인한다. 아니라면 +47로 이동, 폭탄이 터진다. +34에서 eax에 (rsp+0x14)를 옮긴다. 이 값은 처음으로 입력한 정수이다. 이후 eax를 test하는데 eax가 음수이면 폭탄이 터진다. 따라서 eax는 양수이다. 이후 +42에서 eax와 0xe를 비교한다. Eax가 더 작거나 같아야 폭탄이 터지지 않는다. 즉, 처음으로 입력한 정수는 14 이하여야 한다. 이후 edi에 (rsp+0x14)를 옮긴다. 앞서 말했듯이 이 값은 처음 입력한 정수이다. edx에는 0xe를, esi에는 0을 옮긴다. 이후 func4를 호출한다.

Func4에서 값이 반환된 뒤, +71에서는 eax와 3을 비교한다. 만약 다르다면 +83으로 이동하고 폭탄이 터진다. 즉, eax는 3을 반환해야 한다. 또한 3과 (rsp+0x10)을 비교한다. 이 값은 두 번째로 입력한 정수이다. 둘이 같아야 폭탄이 터지지 않고 마무리된다. 즉, 두 번째 입력 값은 3이다.

우리는 func4를 분석해도 좋지만 첫째 입력값이 0~14의 15가지이고 두 번째 정수는 3으로 확정된 상태이다. Brute-force를 이용해도 될 것이다. 그리고 func4는 뜯어보게 되면 재귀 함수이고 간단하지도 않기 때문에 brute-force를 이용하는 것이 현명할 것이다. 진행하면, 올바른 입력 값은 **12 3** 임을 알 수 있다.

* **phase\_5**



+8에서 함수 string\_length를 호출하고 여기서 반환된 값인 eax를 6과 비교한다. 같아야 폭탄이 터지지 않는다. 간단히 이 함수는 입력하는 문자열의 길이를 반환한다고 유추할 수 있다. 즉, 입력 값은 길이 6인 문자열이다. +23에서 edx에 0을 옮긴다. 이후 rsp를 rdi에 옮긴다.

+31에서 Rax에 (rdx+rbx)를 옮긴다. 간단히 생각하면 +5에서 rdi를 rbx에 옮겼는데, 이는 입력한 문자열의 주소를 rbx에 옮겼다고 생각할 수 있다. 즉, +31에서 한 것은 rax에 문자열의 주소에 있는 문자열을 옮긴 것이다. 이후 eax와 0xf를 and연산을 취해 eax의 마지막 4bit만 남긴다. 다음으로 eax에 (rax+0x4024a0)을 넣는다. 즉, 0x4024a0 근처의 값들을 뜯어봐야 한다는 의미이다.



0x4024a0을 x/s를 이용해 열어보면 maduiersnfotvbyl%d %d %s 라는 결과가 나온다. m부터 l까지는 전부 16개의 알파벳으로 구성되어 있다. 즉, rax에 있던 값만큼 문자열을 이동한 위치에 있는 알파벳을 eax에 옮기는 것이다.

+46에서는 rdx+rdi에 al을 옮긴다. 원래 rdi에는 rsp가 있었으므로 rsp+rdx의 주소에 al, 즉 앞서 옮겼던 알파벳을 옮기는 것이다. 이후 rdx의 값을 1 올리고, 6이 아니면 +31로 돌아간다. 즉, 6번 도는 루프이다. 이 루프는 입력한 문자열을 하나하나 떼어서 그 값만큼 0x4024a0에 있는 문자열을 이동해 해당 위치의 알파벳을 rsp와 그 근처 주소에 하나하나 옮기는 것이다. 이렇게 새롭게 완성된 문자열이 rsp 근처에 있다.

루프를 나와서 +59에서 rsp+6에 0을 옮긴다. 이는 NULL을 넣어 문자열을 끝내기 위함이다. 다음으로 esi에 특정 0x402444를 넣는데 이 값을 뜯어보면 flyers라는 문장이 나온다.



이후 두 문자열을 비교해 같아야 폭탄이 터지지 않는다. 이는 우리가 만들어야 하는 문자열이 flyers임을 의미한다. Flyers는 해당 알파벳이 0을 처음으로 한다면 순서대로 0x9, 0xf, 0xe, 0x5, 0x6, 0x7번째에 있다. 여기서 주의할 점은 문자열이므로 우리가 입력하는 문자열 역시 아스키 코드를 보고 참고해서 해야 함을 의미한다. 아스키 코드표를 보면 숫자의 경우 문자로서의 숫자가 16진수로 변환한 숫자의 앞 4bit가 일치한다. 따라서 숫자는 그냥 써도 무방하며 알파벳 중 적당히 16진수로 표현 시 f와 e가 첫 4bit에 있는 것을 찾으면 된다. o, n 등이 있다. 정리하면, 우리가 입력해야 하는 문자열은 **9on567** 이다.

* **phase\_6**

phase 6는 굉장히 복잡한 구조를 가진다. 간단히 요약하면 처음에 이중루프가 있어서 입력 값을 검사하는데 입력 값은 정수 6개이며, 모두 6 이하인 동시에 달라야 한다. 즉, 1~6의 순열이라고 볼 수 있다. 이게 한 부분이고, 다른 부분은 이 6개의 숫자를 배열하는 조건을 다룬다. 숫자에 따라 특정 주소의 값을 받는데 이 값 역시 주소이고, 그 주소로 가면 비로소 어떤 정수가 나온다. 이 정수가 작은 것부터 큰 것까지 올 수 있게 하는 순서를 찾아 입력하면 된다. 좀 더 자세히 설명해보자.

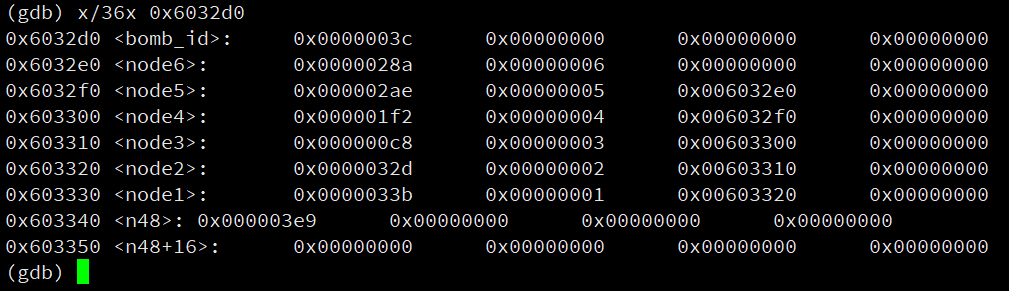
+14에서 rsi에 rsp+0x30인 주소를 넣는다. 즉, rsp+0x30부터 순서대로 입력값이 저장됨을 의미한다. +19에서 read\_six\_numbers가 호출되는 것으로 보아 입력 값은 6개의 정수이다. 이후 순서대로, r13에 0을 넣는다. R14에 첫째 정수의 주소를 넣는다. R15에 r14의 값을 옮긴다. R12에 r13(0)을 옮긴다. Eax에 (r14+4\*r13)을 옮긴다. 즉, 처음에는 eax에 (첫째 정수의 주소+4\*0) => 첫째 정수가 옮겨진다. 이후 eax에서 1을 빼고, 5와 비교해서 작거나 같아야 폭탄이 터지지 않는다. Jbe로 비교했으므로 unsigned int이다. 즉, 첫째 입력 값은 1~6 사이의 정수이다.

이후 +58에서 ebx에 r13(0)+1을 옮긴 후, +62에서 r13과 5를 비교해 같으면 +242로 이동한다. 일단 넘어가면 +72에서 rbp에 r15를 옮긴다. 이 때 r15는 r14와 동일한 값이고, 현재는 첫째 정수의 주소이다. 이후 +75에서 rdx에 ebx를 옮긴다. 현재 ebx에는 r13+1=1이 있다. 이후 +78에서 eax에 (rbp+4\*r12)를 옮긴다. R12에는 r13이 복사되어 0이 있고, rbp에는 r15, 현재는 첫째 정수의 주소가 있다. 즉, 이 과정을 거치면 eax에는 첫째 정수가 복사된다. +83에서 eax와 (rsp+0x30+4\*rdx)를 비교한다. 앞서 rsp+0x30이 첫 정수의 주소였으므로 (첫 정수 주소+4\*rdx) => (첫 정수 주소+4) => 둘째 정수를 비교하는 것이다. 첫째 정수와 둘째 정수를 비교해 달라야 폭탄이 터지지 않는다. 이후 ebx에 1을 더하고 ebx가 5보다 작거나 같으면 +75로 돌아간다. 즉, ebx가 5가 될 때까지 루프를 돌리는 것이다. 이 루프에서는 첫 정수와 다른 2~6번째 정수들의 값이 서로 같은지를 판별한다.

이후 루프를 나와 +102에서 r13에 1을 더하고 +38로 올라간다. 이 큰 루프를 빠져나갈 방법은 +62에서 r13이 5인 상태로 +242로 나가는 것이다. 이는 +106까지의 코드가 이중 루프임을 의미한다. 이 이중루프 내에서는 정수들이 모우 1~6 범위인 동시에 모든 정수가 서로 다른지를 판별한다. 두 번째 큰 루프에서는 두 번째 정수와 나머지 3~6번째 정수가 같은지 판별한다.

이후 루프를 빠져나가 +242로 가면 esi에 0x603330을 옮긴다. 0x603330에 있는 값은 0x33b이다. 다음으로 edx에 1을 옮기고, ecx에 0을 옮긴 후 +115로 이동한다.

+115에서는 rax에 ecx(현재 0)를 넣는다. +118에서 edx와 (rsp+0x30+4\*rax)를 비교한다. (rsp+0x30+4\*rax) => (첫째 정수 주소+4\*ecx) =>첫째 정수이다. 첫째 정수와 edx를 비교해 edx가 더 작으면 +108로 올라간다. 이것도 하나의 루프이다. 루프 내에서는 +108에서 rsi에 (rsi+8)을 옮긴다. rsi에는 기존에 주소인 0x603330가 있었다. 이 근처의 값을 보도록 하자.



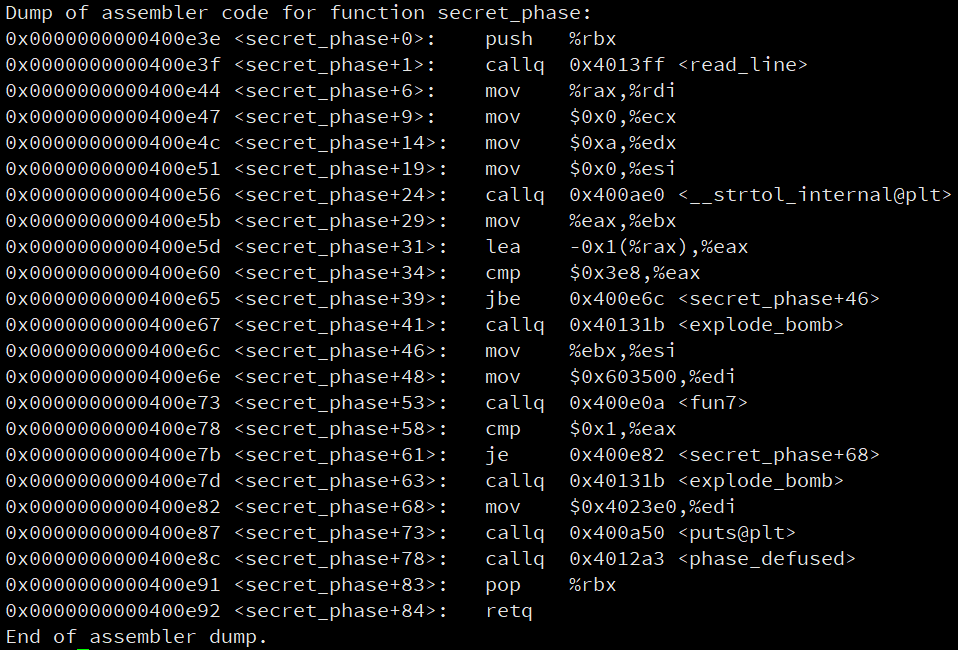
몇 번의 시행착오를 거치면 0x603330의 위아래에 다음과 같은 값들이 있음을 알 수 있다. 우리에게 유의미한 값은 0x6032e0~0x60333c이다. 다시 돌아와서, +108에서는 rsi에 (rsi+8)을 옮기고 rsi에는 0x603330가 있었다. 0x603338에는 또 다른 주소인 0x603320이 있다. 즉, 루프를 한 번 돌면 rsi에 저장되는 값은 0x10씩 감소한다고 볼 수 있다. +112에서는 edx에 1을 더한다. 이후 일련의 과정을 거쳐 edx가 첫째 입력 값과 같을 때까지 루프를 돌린다. 정확히는 첫째 입력값이 1과 차이가 나는 만큼 루프를 돌리고, 그 횟수만큼 rsi는 0x603330에서 0x10씩 감소한다.

루프를 나가면 +124에서 rsi를 rsp+8\*rax인 주소에 옮긴다. 현재는 rax에 ecx의 값인 0이 있었다. 즉, rsi를 rsp에 옮긴다고 생각할 수 있다. 이후 +128에서 ecx에 1을 더하고 5와 비교해 ecx가 더 크면 +148로 점프한다. 그렇지 않으면 다시 esi에 0x603330을 넣고 edx에 1을 더한 뒤 +115도 복귀한다. 역시 이중 루프인데, 두 번째에는 다 같지만 첫째 정수였던 것들이 둘째 정수로 바뀌고, +124에서 rsi를 rsp+8\*n의 위치에 옮긴다는 차이가 있다. 즉, 루프를 돌면서 차례대로 rsp, rsp+8, rsp+0x10, … rsp+0x28까지 0x603XXX를 옮긴다.

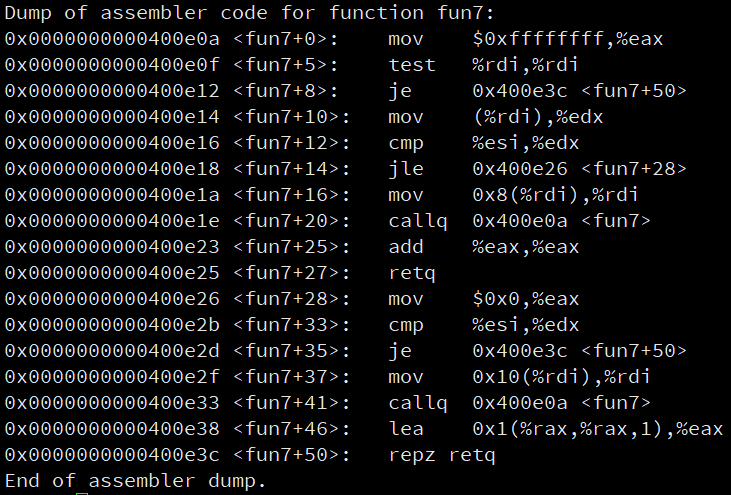
두 번째의 큰 루프를 빠져나가면 +148로 점프한다. +148부터 +208까지는 단순히 앞서 저장한 데이터를 옮기는 것이다. 우리는 rsp에 0x603XXX에 해당하는 주소가 있음을 안다. 먼저 rsp에 저장되어 있던 주소를 받고, 이 주소에 0x8을 더한 위치에 rsp+0x8에 있던 주소를 옮기고, 이 rsp+0x8에 있던 주소에 0x8을 더한 위치에 rsp+0x10에 있던 주소를 옮기고…를 반복하는 것이다. 즉, 0x603XX8의 위치에 주소들을 저장하는 것이다. +205에서 rbx에 rcx를 옮긴다. 이 값은 rsp에 있던 값이다. 이후 ebp에 0을 넣고 rdx에 rbx+0x8, 즉 rsp+0x8에 있던 값을 넣는다. 이후 +219에서 rsp에 있던 주소에 있는 정수와 rsp+0x8에 있던 주소에 있는 정수를 비교한다. rsp에 있던 주소에 있는 정수가 작거나 같아야 폭탄이 터지지 않는다. 이후 ebp의 값을 1 올리고 5와 비교해 값이 다르면 +213으로 올라간다. 이 루프는 검증 루프로, 오름차순으로 정렬이 되어 있는지를 확인하는 작업이다. 0x603XX0에 있는 정수는 0x28a, 0x2ae, 0x1f2, 0xc8, 0x32d, 0x33b이다. 이를 오름차순 정렬하면 0xc8 < 0x1f2 < 0x28a < 0x2f0 < 0x32d < 0x33b 이다. 이에 해당하는 주소를 XXX 부분만 적으면 310, 300, 2e0, 2f0, 320, 330 순이며 앞서 입력 정수만큼 0x10씩 깎이므로, 이를 입력 정수로 다시 변환하면 우리가 입력해야 하는 정수는 **3 4 6 5 2 1** 이 된다.

* **secret\_phase**

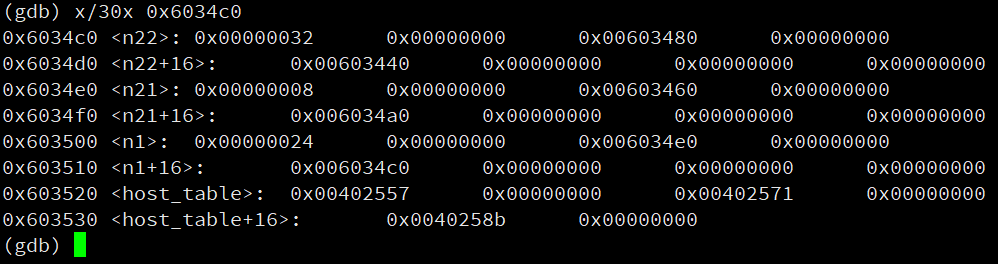
secret phase는 진입 방법이 중요하다. 먼저 secret phase의 트리거링은 phase\_defused 함수에서 일어난다. +5에서 입력한 줄 수가 6이 되어야 진입할 수 있게 했다. 이후 +32에서 0x4024b0을 esi에 옮기는데 이 값을 뜯어보면 %d %d %s이다. 또한 +37에서 0x603a50를 edi에 옮긴다. 이를 뜯어보면 우리가 phase 4에서 입력했던 값임을 알 수 있다. 즉, secret phase에 진입하려면 phase 4의 답 뒤에 문자열을 하나 더 입력해야 한다. 이 때 phase 4에서는 입력 값을 정수 2개로 제한했기 때문에 뒤에 별도 문자열이 있어도 sscanf가 신경쓰지 않는다. 즉, 폭탄이 터지지 않는다는 의미이다. Phase 4의 뒤에 문자열을 입력하면 +55를 통과해 +57에 온다. +57에서는 esi에 0x4024b9를 옮기는데 이 값을 뜯으면 **DrEvil** 이라는 문자열이 나온다. 우리가 입력한 문자열이 다르면 그대로 끝나지만, 이 문자열을 phase 4의 뒤에 추가했다면 secret phase로 진입한다.



Secret phase에서는 분석을 해보면 +53에서 fun7이라는 함수가 호출되고 이 반환값이 1이어야 폭탄이 터지지 않음을 +58에서 확인 가능하다. 찬찬히 변수를 위부터 보자. 값을 하나 추가로 입력받으며 이를 rdi에 저장한다. Ecx에 0을, edx에 0xa를, esi에 0을 저장한다. 그리고 ebx에 입력값 eax를 저장, eax에서 1을 뺀다. 이를 0x3e8과 비교해 eax가 더 작거나 같아야 폭탄이 터지지 않는다. 즉, 입력 허용범위는 1~0x3e9(십진수로 1001)이다. 1000개의 가능한 입력값이 있지만 많아서 brute-force는 사용하지 않는다. 이후 ebx를 esi에 옮기고 edi에 주소 0x603500을 옮긴다. 이 주소에는 36이라는 정수가 있다.

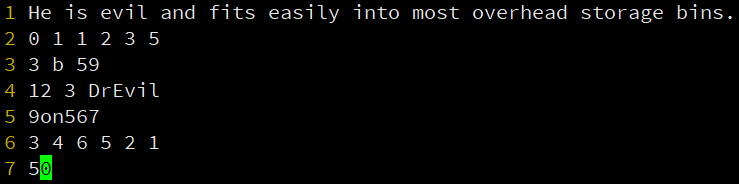


Fun7 함수는 재귀함수이다. 우리는 반환 값이 1이어야 함을 알고 있다. 먼저 +0에서 eax에 0xffffffff를 옮기고 +10으로 넘어가서 edx에 (rdi)를 넣는다. 이는 36을 넣음을 의미한다. 그리고 esi와 비교한다. 이는 esi와 36을 비교한다는 의미이고, esi가 더 크면 +28로 넘어간다. 만일 입력값인 esi가 28 이하인 경우, +27에서 어쨌든 끝이 나는데 이 때 반환값을 보면 무조건 짝수이므로 1이 될 수 없다. 즉, 입력값은 36보다 커야 한다. +28로 가면 eax에 0을 넣고 입력값인 esi와 36인 edx를 비교, 같으면 그대로 끝낸다. 하지만 이러면 0이 반환되므로 같으면 안 된다. +37에서는 (rdi+0x10)을 rdi에 옮긴다. 이는 0x603500 근처 데이터를 뜯어서 확인 가능하다.

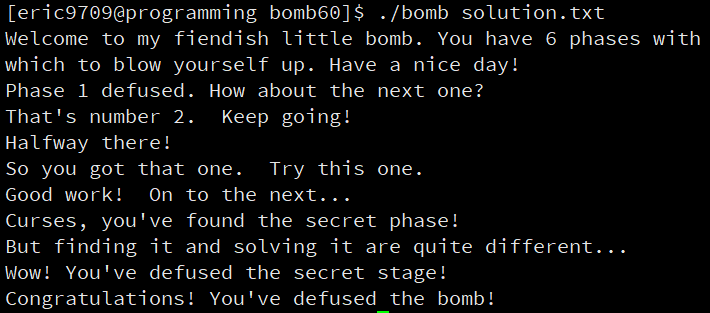


즉, 0x6034c0이 옮겨짐을 알 수 있다. 이후 다시 fun7을 호출한다. 이후 이 함수의 반환값을 받아 2를 곱하고 1을 더해 반환한다. 이렇게 1을 만들려면 두 번째로 호출된 fun7에서 0을 반환, 1+2\*0=1을 반환해야 한다. 즉, 두 번째 fun7에서는 입력값 esi가 0x6034c0에 있는 값과 같아야 한다. 이 값은 0x32, 십진수로는 50이다. 즉, **50**을 입력하면 secret phase가 풀린다.

위 내용을 모두 종합하면 다음과 같은 결과를 얻을 수 있다.



이는 solution.txt의 모습이다.



폭탄을 해체한 모습이다.