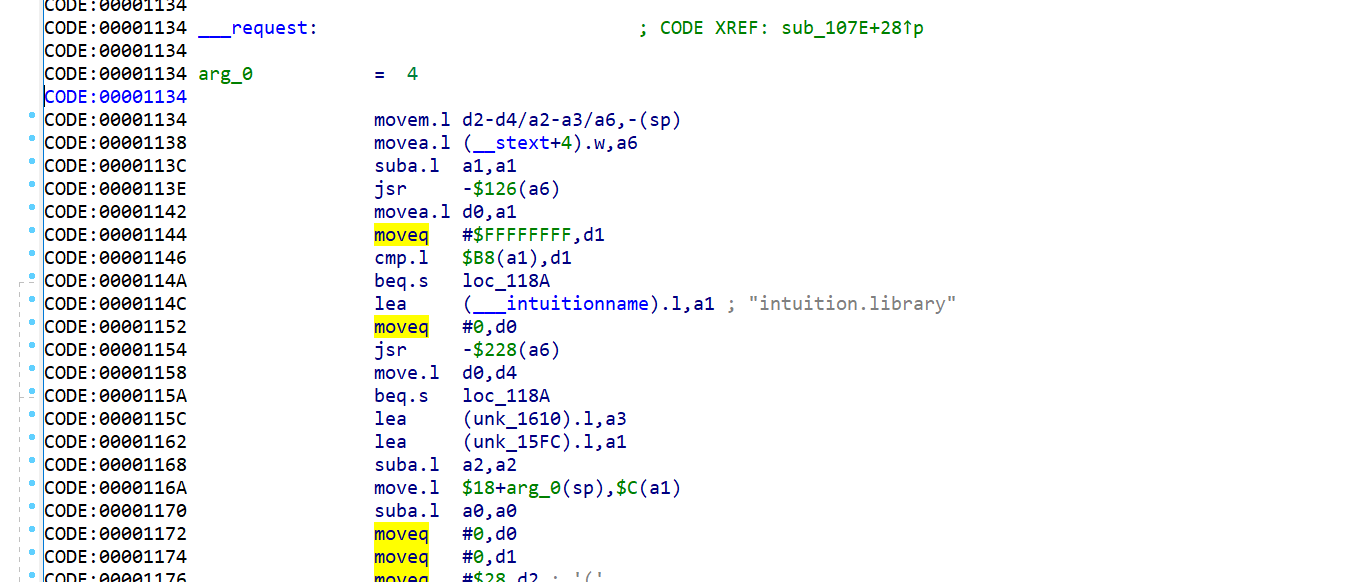
1. 문제 확인



1. 과정

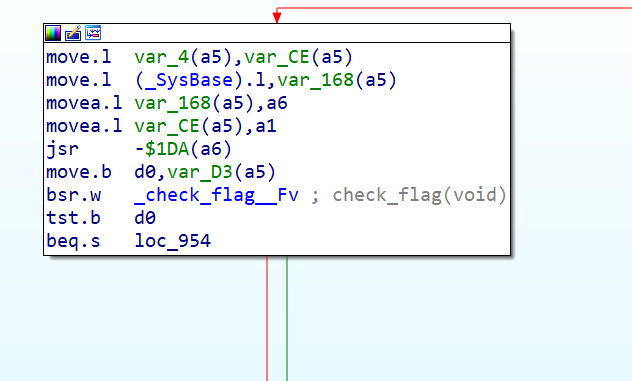
문제는 다음처럼 리버싱 엔지리어링 문제고 AmigaOS에서 실행해야 하는 것 같다.

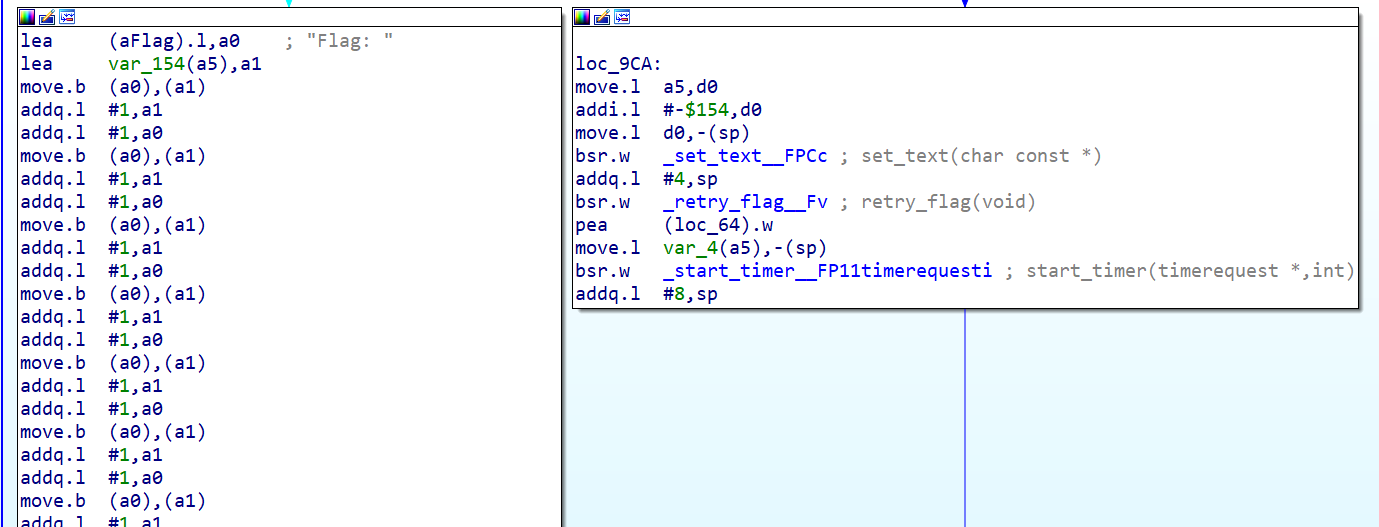
일단 당장 운영체제를 어떻게 시도해야 할지 몰라서 그냥 파일을 먼저 IDA로 열어봤다.



코드는 다음과 같았고 디컴파일 되지 않는 걸로 보아 어떤 조치를 취해야할 것 같다.

우선 조금 더 살펴보았다.





뭔가 검사하는? 부분이 있는 것 같다.

이 코드를 어떻게 하면 좋을까 생각하다가 다른 사람의 라업을 참고해봤다.

여기는 기드라의 어떤 플러그를 사용해서 디컴파일할 수 있다고 하는데 아무리 봐도 설치하는 방법을 모르겠어서 일단 디컴파일된 코드를 먼저 들고 와봤다.

#**include** <stdio.h>

#**include** <string.h>

**char** enc\_chx[] = { 0x8B , 0x84 , 0x9A , 0x9B , 0x9A , 0xB1 , 0xD6 , 0xAF , 0x93 , 0xB2 , 0x81 , 0x8C , 0x84 , 0xAB , 0x9D , 0x9C , 0x8E , 0xB9 , 0xB0 , 0xD9 , 0xA8 , 0xA4 , 0x9C , 0x81 , 0x85 , 0xA0 , 0xA6 , 0xB4 , 0x87 , 0x9A , 0xBB , 0x92 , 0x96 , 0xAD , 0x8C , 0xD7 , 0xB0 , 0x8D , 0x97 };

**char** enc\_idx[] = { 0x16 , 0xC , 0x24 , 0x17 , 0x13 , 0x19 , 0x7 , 0x9 , 0xE , 0x23 , 0x5 , 0x1 , 0x18 , 0x21 , 0xD , 0x10 , 0x12 , 0x1F , 0x1A , 0x1E , 0x22 , 0x0 , 0xF , 0xB , 0x8 , 0x15 , 0x11 , 0x2 , 0x1D , 0x1C , 0x26 , 0x3 , 0x4 , 0x25 , 0x14 , 0x20 , 0x6 , 0x1B , 0xA };

**void retry\_flag() {**

**int var\_1, var\_2;**

**int var\_7, var\_8;**

**for (int i=0; i < 31; i++) {**

**var\_7 = rand() % 26;**

**var\_8 = rand() % 26;**

**var\_1 = enc\_idx[var\_7];**

**var\_2 = enc\_chx[var\_7];**

**enc\_idx[var\_7] = enc\_idx[var\_8];**

**enc\_chx[var\_7] = enc\_chx[var\_8];**

**enc\_idx[var\_8] = var\_1;**

**enc\_chx[var\_8] = var\_2;**

**}**

**}**

**int check\_flag() {**

**char var\_1;**

**var\_1 = enc\_idx[0];**

**if (var\_1 & 1 != 0)**

**return 0;**

**for (int i=2; i < 39; i += 2) {**

**if (enc\_idx[i] < var\_1)**

**return 0;**

**var\_1 = enc\_idx[i];**

**if (var\_1 & 1 != 0)**

**return 0;**

**}**

**var\_1 = enc\_idx[1];**

**if (var\_1 & 1 == 0)**

**return 0;**

**for (int i=3; i < 39; i += 2) {**

**if (enc\_idx[i] > var\_1)**

**return 0;**

**var\_1 = enc\_idx[i];**

**if (var\_1 & 1 == 0)**

**return 0;**

**}**

**return 1;**

**}**

**void decrypt\_flag(char\* dest) {**

**char flag[0x64] = {0};**

**for (int i=0; i < 39; i++) {**

**flag[i] = (~enc\_chx[i]) ^ i;**

**}**

**strcpy(dest, flag);**

**}**

코드는 이러했다. 우선 check\_flag를 살펴보면 인덱스 증가 부분이 +2로 되어있는 것을 확인할 수 있고 짝수 인덱스와 홀수 인덱스를 확인하는데 변수 var\_1에 들어있는 값이 홀수 인덱스 값보다 작은지, 짝수 인덱스보다 더 큰지 검증하는 코드였다. 중간 조건절을 활용해 아닌 것들은 종료시키고 모든 조건을 잘 피한다면 decrupt\_flag를 실행할 수 있는 것 같다.

여기서 짝수 인덱스의 요소는 규칙이 있고 내림차순이어야 하고 홀수 인덱스의 요소는 홀수이고 오름차순이어야 한다는 것을 발견할 수 있다고 한다.

이제 위의 말대로 enc\_idx의 요소를 정렬해보았다.

-짝수 인덱스의 요소(내림차순): 26, 24, 22, …, 0

-홀수 인덱스의 요소(홀수+오름차순): 01, 03, 05, …, 25

이 배열을 다시 정렬해 표시하면 다음과 같다.

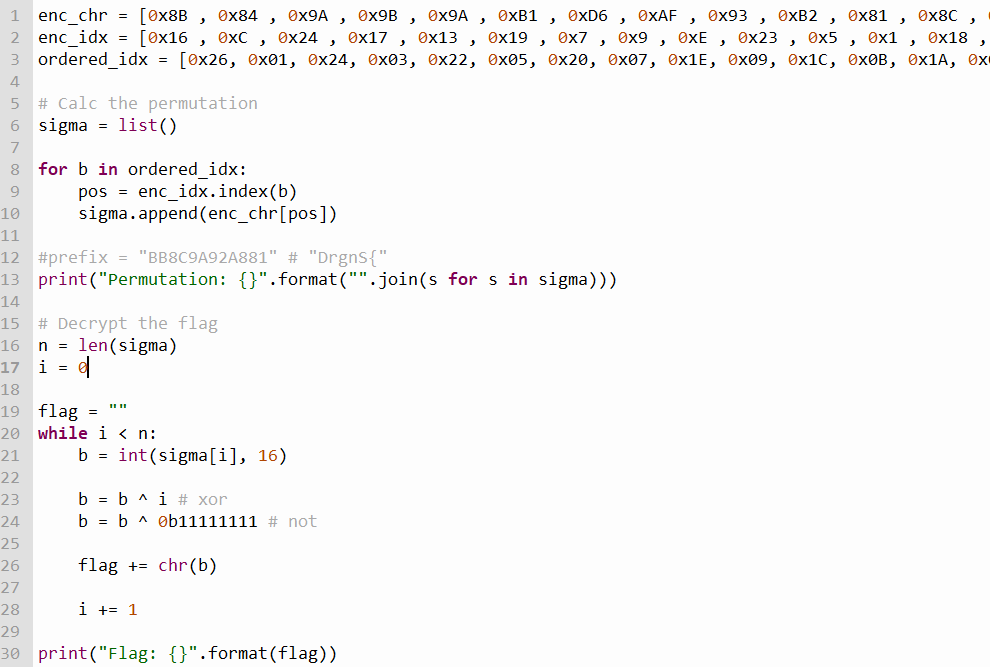
enc\_idx: 26, 01, 24, 03, 22, 05, 20, 07, 1E, 09, 1C, 0B, 1A, 0D, 18, 0F, 16, 11, 14, 13, 12, 15, 10, 17, 0E, 19, 0C, 1B, 0A, 1D, 08, 1F, 06, 21, 04, 23, 02, 25, 00

이제 인덱스는 정렬했고 enc\_chr과 매핑해서 플래그를 구하는 부분을 살펴보겠다.

decrypt\_flag를 확인해보면

**flag[i] = (~enc\_chx[i]) ^ i;**

라고 되어 있는데 이걸 참고해서 파이썬 스크립트를 짰는데 잘 출력이 안되는 부분이 있어서 다른 풀이의 스크립트를 참고해서 플래그를 얻었다.



DragnS{…YouCouldHaveJustWaitedYouKnow}