윈도우즈시스템해킹 (패스워드 크랙)

-양경석(gaeng4@gmail.com)

사용 도구: OLLYDBG(올리디버거)

기본적으로 필요한 단축키
-continue: F9
-ni: f8
-si: f7 (함수 내부까지 들어감)
-소프트웨어 브레이크포인트: F2

1. reverseMe.exe

1) 코드 패칭으로 해결
jnz short 004010D0 을 congratulations가 나오는 곳 위치로 변경.
(jnz 00401205)

jnz --> jmp로 변경
short jmp를 long jmp로 바꾸는 등 여러가지 방법 사용 가능.

마우스 오른쪽 클릭-> copy to excutable -> 마우스 오른쪽 클릭 -> save file

readfile(hfile, buffer,bytestoread,pbytesread,poverlapped);

2) 조건을 만족시킴으로써 크랙

-hFile : 읽고자 하는 파일의 핸들. 이 파일은 GENERIC_READ 액세스 권한으로 열어야 한다.
-lpBuffer : 읽는 데이터를 저장할 버퍼의 포인터, 충분한 길이를 가지고 있어야 한다.
-nNumberOfBytesToRead : 읽고자 하는 바이트 수
-lpNumberOfBytesRead : 실제로 읽은 바이트 수를 리턴받기 위한 출력용 인수. ReadFile은 호출 즉시 이 값을 0으로 만든다. 비동기 입출력을 하지 않을 경우 이 인수는 NULL로 줄 수 없으며 반드시 DWORD형 변수에 대한 포인터를 제공해야 한다.
-lpOverlapped : 비동기 입출력을 위한 OVERLAPPED 구조체의 포인터. 파일을 FILE_FLAG_OVERLAPPED 플래그로 열었으면 이 구조체를 반드시 제공해야 한다. 비동기 입출력을 사용하지 않을 경우 NULL을 주면 된다.

성공하면 0이 아닌 값을 리턴한다. 만약 리턴값이 0이 아닌데 실제 읽은 바이트가 0이라면 파일 포인터가 끝부분(EOF)인 것이다. 실패하면 0을 리턴한다.

문제에서
-hfile: eax
-buffer: reverseM.0040211A
-BytestoRead(70bytes)
-pBytesRead: reverseM.00402173
-pOverlapped: NULL

*00402173에 파일의 바이트수가 입력됨.
cmp al,47 ==> 47=>'G'
글자가 하나 있을때마다 EBX 증가시키며
G가 있을때마다 esi를 증가시키며, esi와 8을 비교함.
G의 갯수가 8개이면 조건 만족시켜 루프를 벗어나게 됨.

createfile // readfile

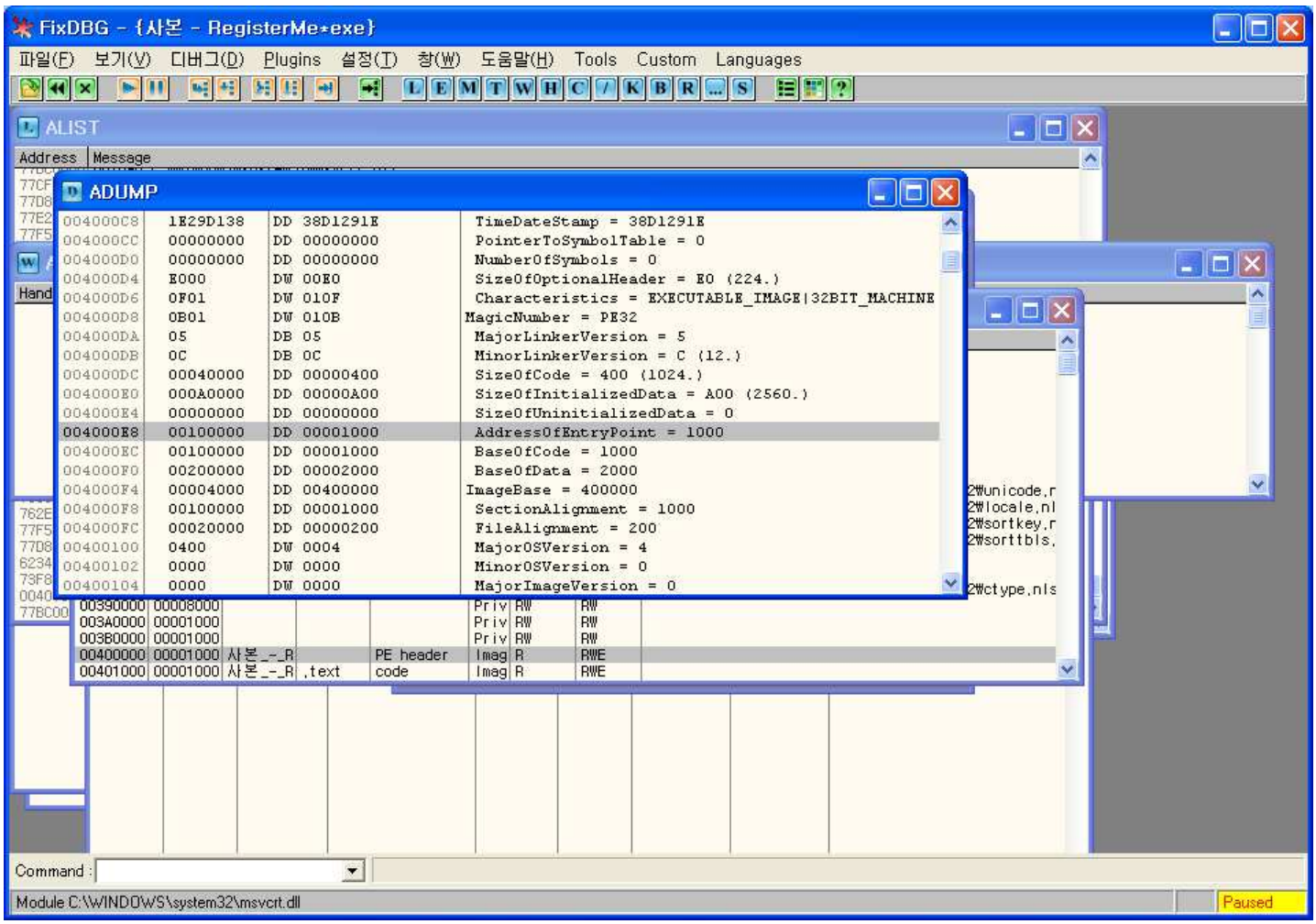
2. register me

처음 실행시 오류메시지

세개의 메시지가 나옴

*nag창 없애기(nag: 귀찮게하다)

M아이콘 눌러보면 00400000 등 주소 보임
00400000눌러보면
4000E8에서 보면



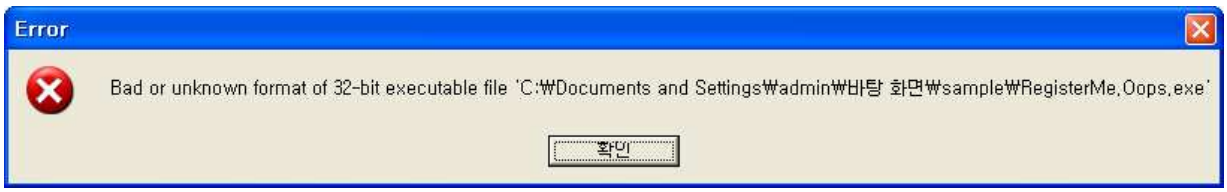
Address of entry point를
1000--> 1024로 변경(시작위치를 처음 메시지창 다음으로 옮김)

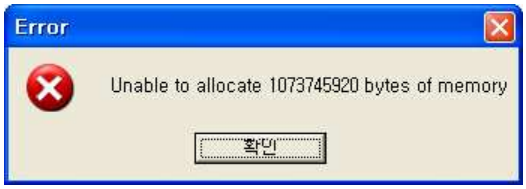
ctrl+G로 주소 이동.

두번째 메시지창 없애기 위해서 NOP으로 채움.

c.f. registerME. WOOPS

register me 파일에 안티디버깅 기법이 적용된 파일.
실행은 되나, 디버거에서 열면 아래와 같은 메시지 뜸.



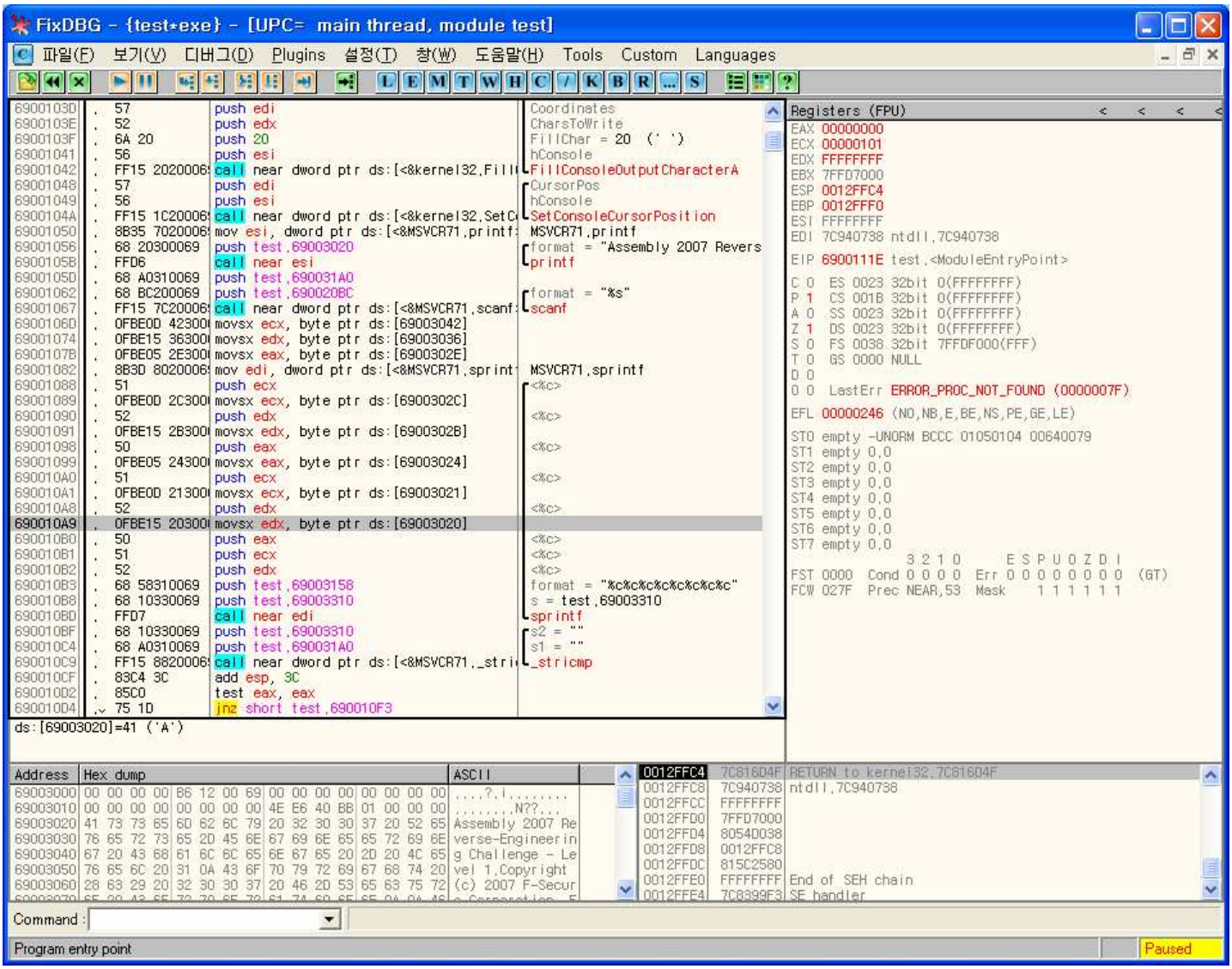


안티디버깅 기법(디버그 하지 못하도록)

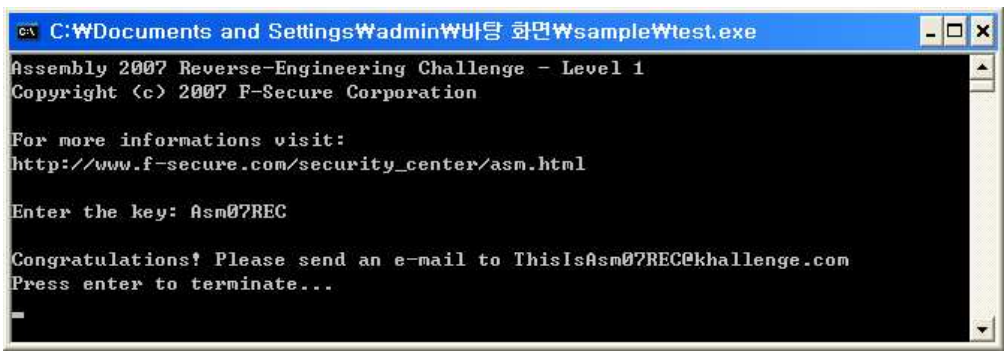
올리디버거에서는 수정후 저장이 안되므로 HxD에서 registerME파일과 비교해가면서 파일 수정하기.
(수정이 성공하면 올리디버거에서 오류없이 로드 가능)

3. test.exe

올리디버거로 열어보니 수상한 부분 발견



%c있는 곳에 눌러보니 아스키 코드가 하나씩 있음
역순으로 넣어보면 << Asm07REC >>



성공(대소문자 상관은없는듯)

리눅스: _startup함수-> main함수-> sub function ...
-->코드를 다 보고 찾아내는 게 좋음
윈도우즈: startup code--> main함수
--> 필요한 부분만 보고 찾아도 됨..

4. abexcm3.exe

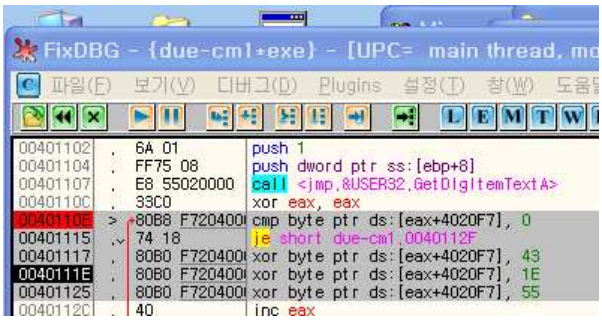
keyfile이름이 abex.l2c 이고 크기가 18바이트이면 됨.
keyfile에 "A" x 18 채우면 끝

5. Crackme_05.exe

오류메시지 앞에 브레이크 걸어두고 실행하면
string1에 시리얼넘버 보여짐.

L2C-57816784-ABEX

6. due-cm1.exe



xor연산을 3번 한 값과 아래의 값과 비교하는 과정을 거침

```
004020D3  7B 61 65 78 64 6D 26 6B 7A 69 6B 63 65 6D 26 3C  {aexdm&kzikcem&<
004020E3  26 66 6D 7F 6A 61 6D 7B 26 6A 71 26 6C 7D 6D 64  &fmjam{&jq&l}md
004020F3  61 7B 7C 00                                      a{|.
```

3번 xor연산 후에 아래와 같은 아스키 문자가 나오도록 해야하므로
프로그래밍을 통해 그 값을 구해보도록 함.
{aexdm&kzikcem&<&fmjam{&jq&l}mda{|.

* 시행착오
char ch[]="{aexdm&kzikcem&<&fmjam{&jq&l}mda{|."
이런식으로 문자를 통해 xor연산을 거쳤더니
simple.crackme.4.nbies.by.duelist
ew가 누락됨. --> 알아보니 0x7F가 아스키 코드로 DEL이므로 문자로 표현이 불가함.

어쩔수없이 아스키코드를 각각 배열에 집어넣어 준 후 연산을 하도록 함.

*c언어 코드

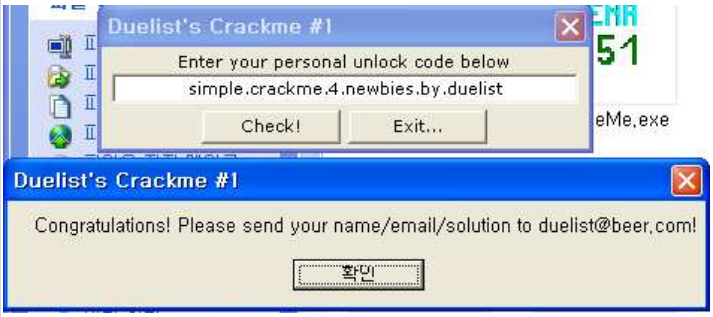
```
#include <stdio.h>
#include <string.h>


int main(){
int i,i_1,i_2,i_3,j,leng;
char ch[]={0x7B,0x61,0x65,0x78,0x64,0x6D,0x26,0x6B,0x7A,0x69,
          0x6B,0x63,0x65,0x6D,0x26,0x3C,0x26,0x66,0x6D,0x7F,
          0x6A,0x61,0x6D,0x7B,0x26,0x6A,0x71,0x26,0x6C,0x7D,
          0x6D,0x64,0x61,0x7B,0x7C,0x00};


leng=strlen(ch);
printf("==============crack tool============\n");
printf("length: %d\n",leng);

for(j=0;j<leng;j++){
for(i=0x00;i<0xff;i++)
{
  i_1=i^0x43;
  i_2=i_1^0x1E;
  i_3=i_2^0x55;
  if(i_3==ch[j]){printf("%c",i); break;}
}
}
printf("\n");
```

```
printf("====================================\n");
printf("                    -HACKED\n");

return 0;
}
```

**a xor 43 = 7B 인 경우 a= 43 xor 7B 이렇게 해도 됨. (교환법칙 성립)

<결과>

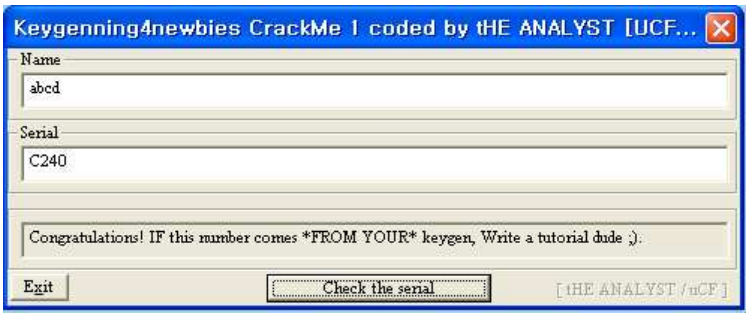
```
./findhex
=============crack tool============
simple.crackme.4.newbies.by.duelist
====================================
                                -HACKED
[root@kh /root]#
```



성공!

7. k4n.exe

name 4~50글자
중간에 브레이크포인트 잡아두고 name을 임의로 abcd라고 입력해줌.
string2에 C240이 뜨고 비교하는 루틴 있음
serial에 C240써줌
완료



난이도에 비해 방향 잡는데 오래 걸림.

8. due-cm2.exe

디버거로 코드 분석 결과
1. due-cm2.dat 생성
2. 파일사이즈 0x12(18)바이트 이상
3. due-cm2.dat 파일 체크

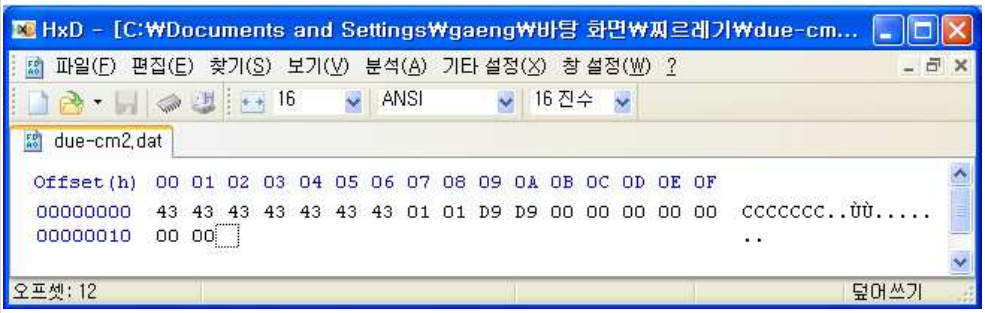
0x00인 경우 esi 2인지 체크(작은 경우 종료, 큰 경우 continue)
0x00아닌 경우: 0x01인지 체크 (0x01이면 esi, ebx증가 / 아니면 ebx증가) --> 0x01이 2개 이상 있어야 함

<esi 2이상 통과>

esi, ebx 초기화

파일내의 hex판단
-0인경우: esi가 0x1D5(469)인지 판단(469=7x67(0x43))
-1인 경우: 상동
-나머지: 다 더한 값이 0x1D5인지 판단
(0이나 1인 경우는 esi가 0x1D5가 안나오므로 "나머지" 케이스로 가야함)

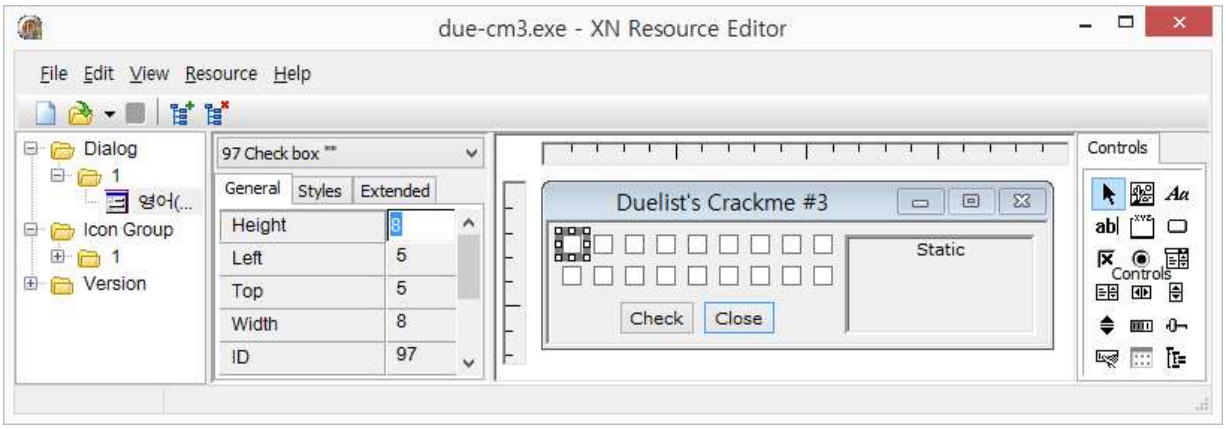
<esi: 1D5조건 통과>
0x01인지 판단 후, esi가 0x1B2(434)인지 판단
2로 나누면 217(0xD9, 2개면 됨)



성공-꽤 오래 걸렸음..

9. due-cm3.exe

소스를 보다보니 각 칸이 가리키는 숫자가 뒤죽박죽이다.
힌트에서 나온 것처럼 리소스 에디터(XN 리소스 에디터라는 프로그램 사용)로 열어보니



004020FE

ESI	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	(19)
HEX	16	49	5E	15	27	26	21	25	1D	59	53	37	31	48	5D	0C	61	52	4D

체크박스

체크박스 순서	1-1	1-2	1-3	1-4	1-5	1-6	1-7	1-8	1-9	2-1	2-2	2-3	2-4	2-5	2-6	2-7	2-8	2-9
해당 ESI	17	2	3	1	8	6	7	10	11	4	12	13	14	15	16	18	5	9
HEX	61	49	5E	16	25	26	21	59	53	15	37	31	48	5D	0C	52	27	1D

공식:
[체크박스에 체크한 것의 해당 ESI] * [ESI에 해당하는 HEX] * [ESI+1에 해당하는 HEX]

EAX x ECX x ESI 한 후 지금까지값과 더함.
여기다가 4D곱한 값이 0F35466이 되어야 하므로
0F35466 / 4D = 328FE 이므로 합은 328FE가 되어야 한다.

프로그래밍 해보자~
18 C n의 조합이 나오므로 가짓수가 매우 많다~

아이디어
1. 18개의 선택지 중에서 1개, 2개, 3개 .. 18개를 고른 후 고른 수를 더하기
2. arr[18]을 만들어서 0, 1을 순차적으로 반복하여 올라가도록 한 후 행렬곱 이용
--> 2진수 1111111111111111~000000000000000000 에 해당하는 16진수(10진수)값인

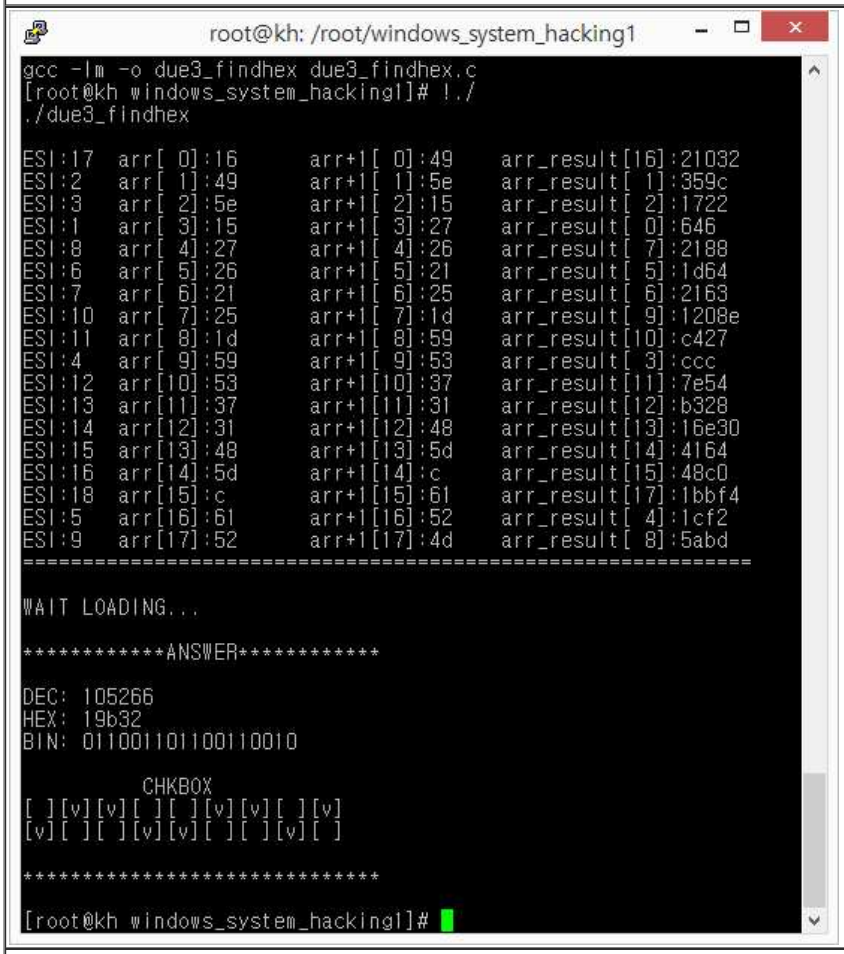
262143(0x3FFFF) ~ 0까지 감소시키면서 arr[18]의 arr[0]~arr[17]에 자릿수별로
0 또는 1이 저장되도록 한 다음에 행렬 곱을 이용.
행렬값이 1인 경우 체크표시 한 경우에 대응되며,
행렬값이 0인 경우 체크표시 하지 않은 경우에 대응되어
위 프로그램의 체크루틴과 동일한 기능을 할 수 있음.

<첫번째 해결방법 프로그래밍>

due3_findhex.c

```
#include <stdio.h>
#include <math.h>
int main(){

int i,j=0,pow_calc,temp,a,sum;
int answer=0x328FE;
int arr_bin[18];

//0x004020FE
int arr1[]={    0x16,0x49,0x5E,
                0x15,0x27,0x26,
                0x21,0x25,0x1D,
                0x59,0x53,0x37,
                0x31,0x48,0x5D,
                0x0C,0x61,0x52,0x4D };

//CHECK BOX -이 배열은 계산시에는 필요없음.
/*
int arr2[]={    0x61,0x49,0x5E,
                0x16,0x25,0x26,
                0x21,0x59,0x53,
                0x15,0x37,0x31,
                0x48,0x5D,0x0C,
                0x52,0x27,0x1D  };
*/

int ecx[]={ 17,2,3,1,8,6,7,10,11,4,12,13,14,15,16,18,5,9};

int arr_result[18]={0,};
printf("\n");
for(i=0;i<18;i++){
arr_result[i]=ecx[i]*arr1[ ecx[i]-1 ]*arr1[ ecx[i]-1+1 ];
printf("ECX:%d\t",ecx[i]);
printf("arr[%2d]:%x\t",i,arr1[i]);
printf("arr+1[%2d]:%x\t",i,arr1[i+1]);
printf("arr_result[%2d]:%x\n",ecx[i]-1,arr_result[i]);
}
printf("================================================================\n\n");

//BINARY CASE
printf("WAIT LOADING...\n");
//0x111111111111111111 = 0x3FFFF = 262143
for(i=0,j=0;i<=262143;i++){ //왼쪽은 작->큰 ,우측은 큰->작 for(i=262143,j=0;i>=0;i--){
arr_bin[0]=i;
//printf("arr_bin[17-j]: %d, i: %d ",arr_bin[17-j],i);
  for(j=17;j>0;j--){
   temp=arr_bin[17-j];
   pow_calc=pow(2,j);
//printf("j: %d 17-j: %d temp: %d ",j,17-j,temp);
   arr_bin[17-j]=temp/pow_calc;
   arr_bin[17-j+1]=temp%pow_calc;
   //printf("arr_bin[%d]=%d (%x)\n",17-j,arr_bin[17-j],arr_bin[17-j]);
   if(17-j+1==17){
    //printf("arr_bin[%d]=%d (%x)\n",17-j+1,arr_bin[17-j+1],arr_bin[17-j+1]);
   }
  }
//printf("\n");

for(j=0,a=0,sum=0;j<18;j++){
  a=arr_bin[j]*arr_result[j];
  sum=sum+a;
  //printf("a: %x\n",a);
  //printf("sum: %x\n",sum);
  }
       if(sum==answer){
        printf("\n**********ANSWER**********\n\n");
        printf("DEC: %d\n",i);
        printf("HEX: %x\n",i);
        printf("BIN: ",i);
         for(j=0;j<18;j++){
              printf("%d",arr_bin[j]);
         }

        printf("\n\n          CHKBOX\n",i);
         for(j=0;j<18;j++){
              if(arr_bin[j]==0){printf("[ ]");}
```

```
            else{printf("[v]");}
            if(j==8){printf("\n");}
        }

        printf("\n\n*****************************\n\n");
       break;
       }
}

return 0;
}
```

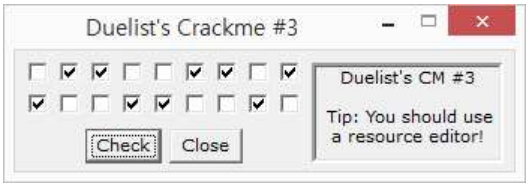


보기 쉽도록 체크박스 그림까지 넣어보았다.
행렬arr[j]을 나눗셈시 직접 이용하면 0으로 인식하는 버그?를 발견한 바
temp=arr[j] 로 넣어준 후 계산 후 돌려주는 방법으로 우회하였다.
2^n을 계산하기 위한 라이브러리함수로 pow(x,n)를 사용하여고 math.h를 인클루드 했고,
리눅스에서 math.h를 인클루드 한 경우 아래와 같이 -lm 옵션을 추가하여야 한다.

#> gcc -lm -o due3_findhex due3_findhex.c

중간에 생각대로 출력이 되지 않아 버그를 발견하는 데 꽤 많은 시간을 할애함.
프로그래밍 작성 후 정답이 나왔으나, 실제로 크랙이 되지 않아 확인해보니
i)처음 사용하였던 리소스해커라는 프로그램이 id값을 잘못 알려주었으며
ii) 바이너리 분석이 조금 잘못되었음을 인지하여 다시 분석함.

많은 고생끝에 성공!



<두번째 해결방법 프로그래밍>
*랜덤함수를 이용하여 푸는 방식을 생각해봄.

a1.c

```
#include <stdio.h>
#include <stdlib.h>

int main(){

int arr[]={0x21032,0x359c,0x1722,0x646,0x2188,0x1d64,
       0x2163,0x1208e,0xc427,0xccc,0x7e54,0xb328,
       0x16e30,0x4164,0x48c0,0x1bbf4,0x1cf2,0x5abd};

int ran[18]={0,};
int i,sum;
int answer=0x328FE;

srand(time(NULL));

while(1){
for(i=0,sum=0;i<18;i++){
  ran[i]=rand()%2;
  sum=sum+arr[i]*ran[i];
}
if(sum==answer){
  printf("\nBINARY\n");
  for(i=0;i<18;i++){ printf("%d",ran[i]); }
  break;
}
else{ sum=0; }
}
  printf("\n\nCHKBOX\n",i);
   for(i=0;i<18;i++){
    if(ran[i]==0){printf("[ ]");}
    else{printf("[v]");}
    if(i==8||i==17){printf("\n");}
   }

return 0;
}
```

코드짜기 훨씬 간단하고, 라인수도 짧아졌음.
문제를 해결하기만 하면 되는 이 문제의 특성상 훨씬 좋은 프로그램이라고 판단됨.

```
./a1

BINARY
011001101100110010

CHKBOX
[ ][v][v][ ][ ][v][v][ ][v]
[v][ ][ ][v][v][ ][ ][v][ ]
```

이 문제에 대해 인터넷에 올라와있는 소스코드 분석
강사님이 주신 코드가 컴파일이 되지 않아 수정하였습니다.

sample.c

```
#include <stdio.h>
char checked[18];
void flag_checked(int i)
{
int j;
    for(j=0; j < 18; j++) {
        if (i & (1<<j))
            checked[j] = 0x31;
        else
            checked[j] = 0x30;
    }
}

int main(){

    int keyvalue[]={0x21032, 0x359C, 0x1722, 0x646, 0x2188, 0x1D64, 0x2163, 0x1208E,
            0xC427, 0xCCC, 0x7E54, 0xB328, 0x16E30, 0x4164, 0x48C0, 0x1BBF4, 0x1CF2, 0x5ABD};
```

```
    int sum = 0;
    int answer = 0x328FE;
    int i,j;

    for(i = 0; i < 262143; i++) {
        sum=0;
        flag_checked(i);

        for(j=0;j<18; j++) {
            if(checked[j]=='1') {
                sum += keyvalue[j];
            }
        }
        if (sum == answer)
            break;
    }

    for(i=0; i < 18; i++) {
        printf("%c", checked[i]);
        if(i == 8||i==17)
            printf("\n");
    }
    return 0;
}
```

결과

1. 들어가며
원래 코드는 문법상 c언어에 맞지 않아 일부 수정 작업을 거쳐 실행가능한 프로그램으로 만들었습니다.
이 프로그램은 18개의 키값 중에서 임의의 키를 선택한 후(key_x, key_y, key_z ...) 이것들의 합이 0x328FE가 되는 x,y,z ... 를 찾습니다.
18개의 키 각각이 체크되거나(1), 체크되지 않거나(0) 하는 것을 2진수의 1과 0으로 표현하여 모든 케이스들을 브루트포싱 할 수 있는 알고리즘을 활용하였습니다.
다만, c언어에서 2진수 자체를 표현이 불가하므로 AND연산과 shift연산을 통하여 10진수를 2진수로 변환하여 이용하는 과정이 포함되어 있으며 이 프로그램의 코드 중에서 가장 핵심이 되는 부분이라 판단됩니다.

2. 코드 분석

코드는 크게 [1] main함수 부분과 [2] flag_checked함수 부분으로 나누어 볼 수 있습니다.
1)메인함수 부분에서는 i)선택된 키값의 합을 구하고, ii)그 합이 0x328FE과 같은 경우 iii)체크되었는지 여부를 출력해주는 간단한 코드로 이루어져 있습니다.
2) 이 프로그램의 핵심인 flag_checked 함수 부분입니다. 2의0승자리부터 2의17승자리까지 1과 의 AND 연산을 통해 체크되었는지 여부를 판단합니다.

```
#include <stdio.h>
char checked[18];
```

[1] flag_checked함수 부분

```
void flag_checked(int i)
{
int j;
    for(j=0; j < 18; j++) {
        if (i & (1<<j))
            checked[j] = 0x31; //0x31는 아스키문자로 1
        else
            checked[j] = 0x30; //0x30는 아스키문자로 0
    }
}
```

<flag_checked함수: 18개 항목 각각이 체크가 되어있는지 여부 판단하는 함수>
if (i & (1<<j)) 부분을 살펴보면,
일단 (1 << j) : 1을 j만큼 좌측으로 shift연산을 합니다.

1비트를 좌측으로 이동시키는 것은 수를x2 한 것과 같은 기능을 합니다.
ex) a=10인 경우, a<<1=20, a<<2=40 ...

따라서 j는 0부터 17까지 변하므로
1<<0, 1<<1, 1<<2 ... 1<<17의 연산을 하게 되며
이는 즉, 각 자리가 1인 18자리의 2진수를 나타냅니다.

예를 들어 111이라는 2진수를 10진수로 변환할 경우에
1 x (2^2) + 1 x (2^1) + 1 x (2^0) =7 과 같이 계산을 합니다.
이 때 (2^2), (2^1), (2^0)은 2진수의 자릿수를 나타내는데,
해당 자릿수가 가진 값이 모두 1이라는 의미입니다.
이를 shift연산으로 표현하게 되면
(1<<2)+(1<<1)+(1<<0) =7 과 같이 된다.

결과적으로 1<<j는
(1<<0), (1<<1), (1<<2) ... , (1<<17) 의 연산을 연속해서 하면서
11 1111 1111 1111 1111 이라는 2진수의 각 자리를 말하게 됩니다.

i & (1<<j)
i와 (1<<j)를 AND연산하고 있습니다.
즉, 둘다 1인 경우에만 1이 되는 AND연산을 통하여
i의 2진수에서의 해당 자릿수가 1인지 구분(MASKING)하고있습니다.

예를들어 10진수212992는 2진수로 표현하면 110100000000000000 입니다.
사람은 10진수로 보지만, 컴퓨터는 2진수로 보기 때문에 수가 10진수인지 16진수인지 2진수인지는 컴퓨터에게는 상관없이 2진수로 연산을 수행합니다.
따라서 10진수 212992를 11 1111 1111 1111 1111과 AND연산을 하게 되면
    11 0100 0000 0000 0000 (우리가 쓰는 10진수 212992를 컴퓨터가 이해하는 형태)
AND 11 1111 1111 1111 1111
---
    11 0100 0000 0000 0000 좌측과 같이 나옵니다.

프로그램상에서는 i값을 0부터 262143까지 증가시키며 18자리의 1과 MASKING을 합니다.

*MASKING과정이 필요한 이유?
c언어에서는 2진수를 표현하는 방법이 따로 존재하지 않습니다.
숫자만 쓰면 10진수로 인식을 하며, 0x를 앞에 붙이면 16진수로 인식을 하며, 0을 앞에 붙이면 8진수로 인식하나, 2진수는 따로 이러한 것을 지원하지 않습니다.
우리는 2진수를 0부터 1씩 증가시켜주는 연산이 필요하나, 이를 직접 표현할 수가 없으므로,
10진수를 피연산자로 입력하여 각 자리수와 1을 AND연산하는 방법으로 c언어가 지원하지 않는 2진수를 우회사용할 수 있습니다.

이 함수는 void형으로써 함수 자체가 리턴해주는 값은 없지만, 전역변수로 선언되어있는 checked[] 배열에
체크가 된 경우 1을, 체크가 되지 않은 경우 0을 넣어줍니다.

[2] main함수 부분

```
int main(){

    int keyvalue[]={ 0x21032, 0x359C,  0x1722, 0x646, 0x2188, 0x1D64, 0x2163, 0x1208E, 0xC427,
                     0xCCC, 0x7E54, 0xB328, 0x16E30, 0x4164,  0x48C0, 0x1BBF4, 0x1CF2,  0x5ABD };
    int sum = 0;
    int answer = 0x328FE;   //선택된 키값들의 합이 최종 일치해야하는 값
    int i,j;

    for(i = 0; i < 262143; i++) {  //262143(10진수) = 0x3FFFF(8진수) = 11111111111111111(2진수)
                                   //i=0부터 i=262143 전까지 증가시키게 되면 0과 1만 가질 수 있는 2진수의 성질상,
                                   //빠짐없이 18개의 키값중에서 선택될 모든 가능성에 대해서 체크할 수 있습니다.
        sum=0;
        flag_checked(i);

        for(j=0;j<18; j++) {
            if(checked[j]=='1') {     //체크가 되어있는 경우 배열에 0x31이라는 아스키문자를 넣어주었기에 문자인 '1'로 조건을 판단합니다.
                sum += keyvalue[j];   //체크가 되어있는 경우에만 키값을 더해줍니다.
            }
        }
        if (sum == answer)    //체크가 되어있는 키값의 합이 answer와 같은 경우 정답을 찾은 경우이므로 for문을 빠져나갑니다.
            break;
    }

    for(i=0; i < 18; i++) {    //출력하는 부분. 체크되어야 할 부분이 1로, 체크되지 않아야 할 부분이 0으로 출력됩니다.
        printf("%c", checked[i]);
        if(i == 8||i==17)
            printf("\n");
    }
    return 0;
}
```

3. 코드 분석을 마치며

같은 고민을 했던 사람의 코드를 보면서, 2진수로 처리하고자 했던 아이디어와 2진수 표현이 불가한 c언어의 문제의식에서 코딩을 전개한 상황은 저와 유사했습니다.
다만, 저는 18개의 배열을 이용하여 각 자릿수에 해당하는 값(0또는 1)을 각 배열에 넣은 후, 각 키값과 곱하는 연산을 통하여 해결한 반면,
이 코드에서는 shift연산을 통한 자릿수 표현과 and연산을 통한masking을 통하여 다소 생각하기 힘든 방법을 통하여 구현하였습니다.
기본적인 아이디어가 비슷하였기에 코드를 분석하는 데에는 어려움이 없었으나, 이 코드를 보기 전에는 생각하지 못한 구현 방법을 알게 되어서 향후 코딩시에 다양한 방법으로 활용될 수 있으리라 판단됩니다.
이번 문제 해결을 위하여 3가지 다른 방법으로 코딩을 해 보았고, 이 코드까지 분석함으로써 같은 문제에 대해서 최소 4가지 이상의 해법을 생각할 수 있었습니다.
4가지의 코드 중에서 항상 어느 것이 좋은 코드라고 할 수 없으므로, 여러가지 코딩 상황에서 최적의 코딩을 선택할 수 있도록
여러가지 알고리즘 구현방법을 생각하는 것 뿐만 아니라, 많은 사례들을 다양하게 접할 수 있는 기회를 갖는 것도 중요하다는 것을 느끼게 되었습니다.
마지막으로 생각을 글로 옮기는 것의 어려움에 대해 다시금 깨닫게 되었습니다. 부족함이 많이 느껴져 배우는 것이 많았던 기회였습니다. 다음번에는 더 좋아질 것이라는 희망을 품으며 마칩니다.