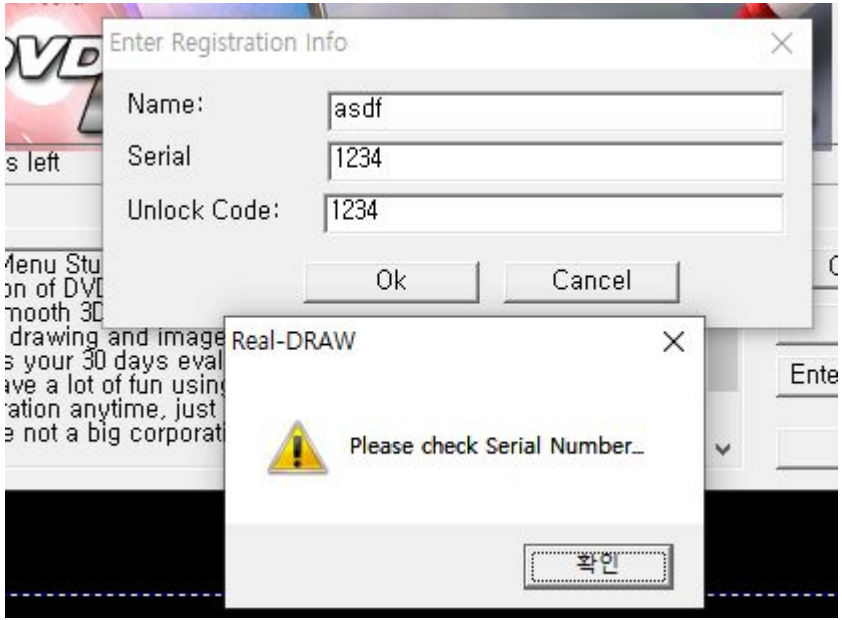
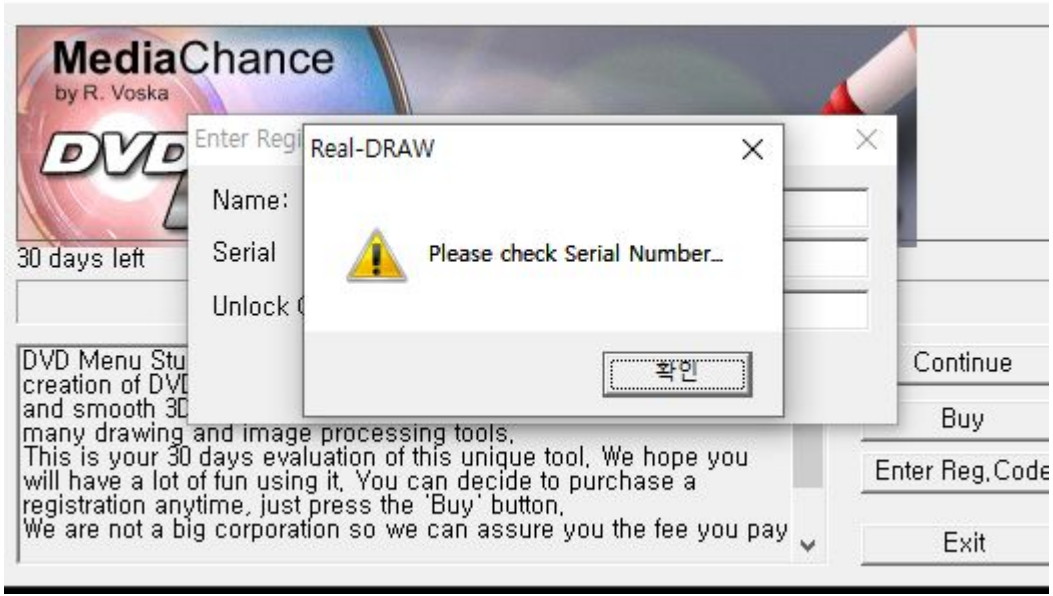
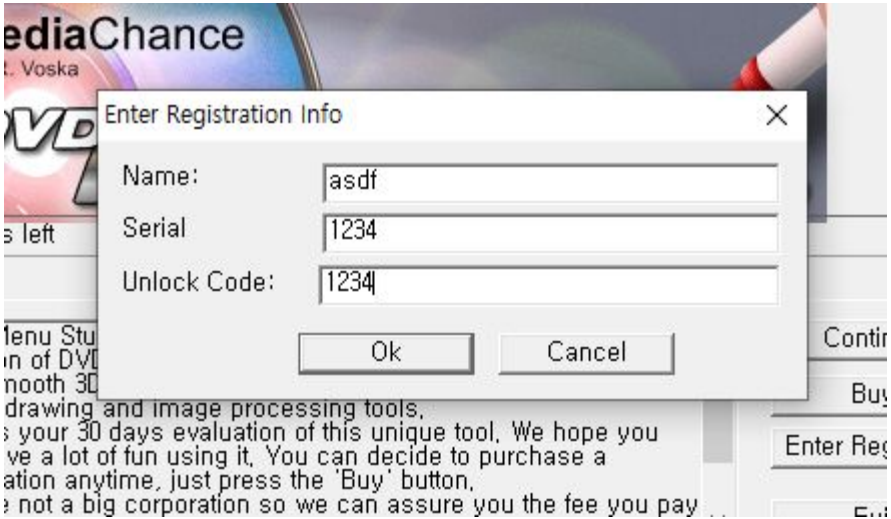


프로그램 시작시 나타나는 화면이다. 30일 남았다는 메시지가 나오고, 등록하라는 화면과 함께 Name, Serial, Unlock Code를 입력하라고 나오고, 시리얼 번호를 잘못 입력시 오류 메시지가 뜬다.



시리얼 코드를 잘못 입력하면, 해당 오류메시지가 뜬다. 문자열을 검색해 해당 영역으로 이동한다.

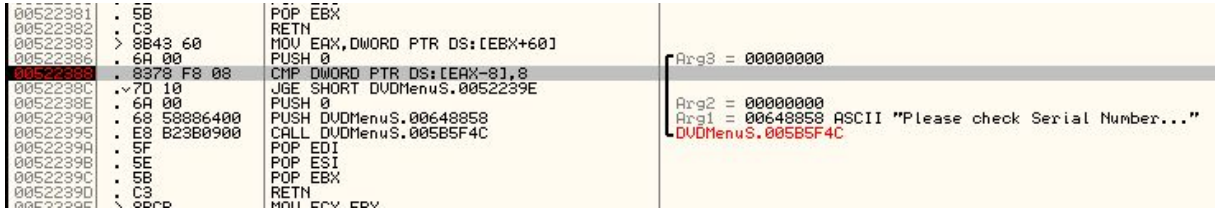



```
00522368  .∨EB 05        JMP SHORT DVDMenuS.0052236F
0052236A  > 1BC0         SBB EAX,EAX
0052236C  . 83D8 FF      SBB EAX,-1
0052236F  > 85C0         TEST EAX,EAX
00522371  .∨75 10        JNZ SHORT DVDMenuS.00522383
00522373  . 50           PUSH EAX                          ; Arg3
00522374  . 50           PUSH EAX                          ; Arg2
00522375  . 68 78886400  PUSH DVDMenuS.00648878            ; Arg1 = 00648878 ASCII "Please enter your name"
0052237A  . E8 CD3B0900  CALL DVDMenuS.005B5F4C            ; DVDMenuS.005B5F4C
0052237F  . 5F           POP EDI
00522380  . 5E           POP ESI
00522381  . 5B           POP EBX
00522382  . C3           RETN
00522383  > 8B43 60      MOV EAX,DWORD PTR DS:[EBX+60]
00522386  . 6A 00        PUSH 0                            ; Arg3 = 00000000
00522388  . 8378 F8 08   CMP DWORD PTR DS:[EAX-8],8
0052238C  .∨7D 10        JGE SHORT DVDMenuS.0052239E
0052238E  . 6A 00        PUSH 0                            ; Arg2 = 00000000
00522390  . 68 58886400  PUSH DVDMenuS.00648858            ; Arg1 = 00648858 ASCII "Please check Serial Number..."
00522395  . E8 B23B0900  CALL DVDMenuS.005B5F4C            ; DVDMenuS.005B5F4C
0052239A  . 5F           POP EDI
0052239B  . 5E           POP ESI
0052239C  . 5B           POP EBX
```

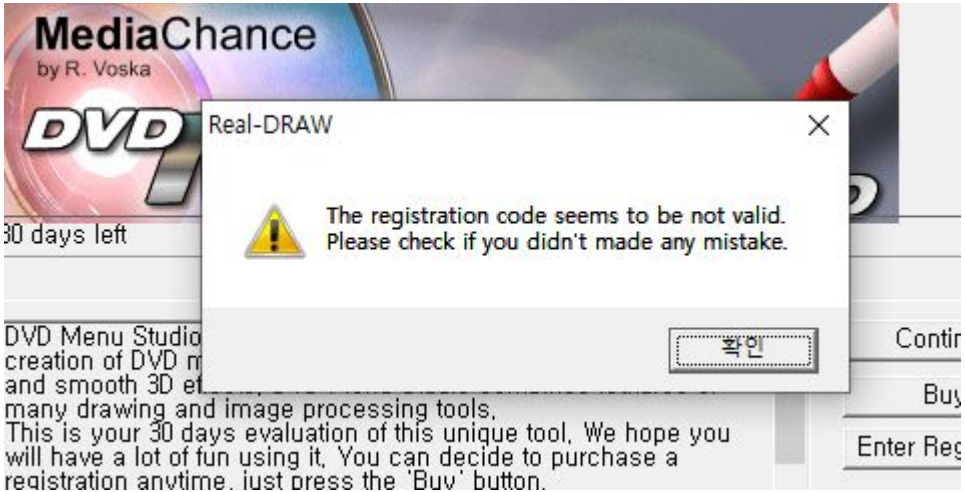
아까 검색한 문자열에 bp를 걸고
필드를 모두 채우고 실행한다.

그러면 해당 영역에서 멈추게된다.

첫번째 문자열은 name 필드를
채우지 않았을 경우 나오는 메시지
같고, 두번째는 시리얼 번호를
틀렸을 경우 나오는 메시지다.



두번째 문자열은 입력한 시리얼 코드의 길이를 비교한다. 8 미만이면 해당 오류를 출력한다. 즉, 시리얼 코드의 길이는 8이상이어야 된다.

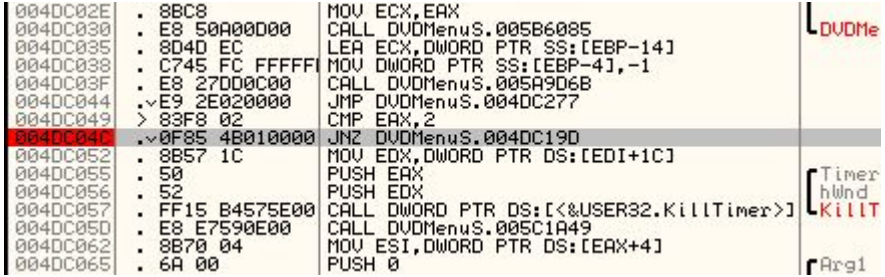


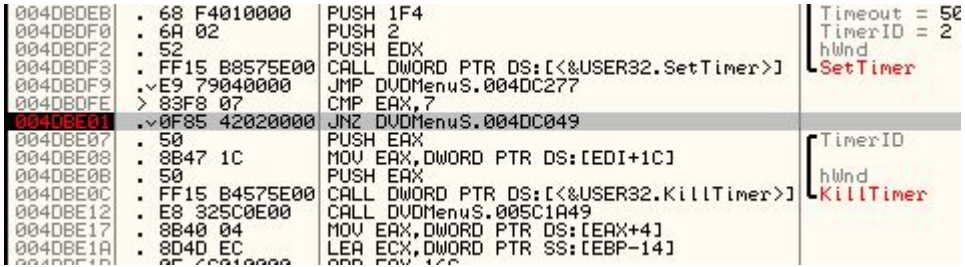
이번에는 시리얼 코드를 8글자 이상 입력하고 실행했더니 시리얼 번호가 맞지 않는다고 나온다. 해당 오류 메시지로 검색을 해본다.

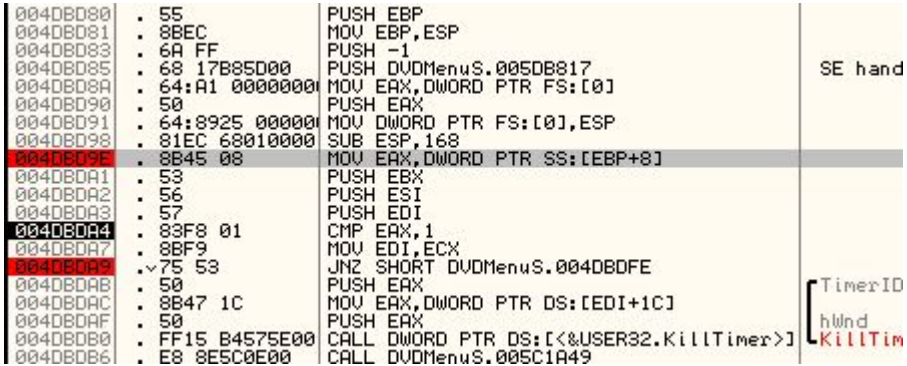
해당 부분에서 멈추게 된다. 두개의 메시지가 보이고, 위에는 시간과 관련된 함수가 보인다.

밑에 두 메시지는 등록 성공/실패시 보이는 메시지다.

4DC1A0부분에서 분기를 결정한다.EAX 값과 비교해 3과 같으면







4DC1AO 함수는 해당 영역에서 분기되는걸 알 수 있다.

또 40C04C는 40BE01 부분에서 분기하는걸 알 수 있다.

이렇게 분기문을 쭉 따라가다 보면 최초로 시작되는 부분을 찾을 수 있다.

4DBD9E에서 EAX 값을 설정하고, 그 뒤로 어떤 값과 비교해 분기를 결정한다.

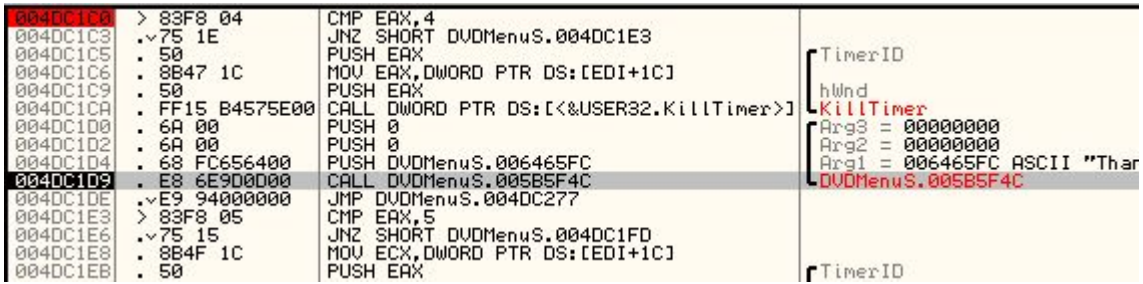
```
004DC1B1  . 68 5C666400     PUSH DVDMenuS.0064665C
004DC1B6  . E8 919D0D00     CALL DVDMenuS.005B5F4C
004DC1BB  .~E9 B7000000     JMP DVDMenuS.004DC277
004DC1C0  > 83F8 04         CMP EAX,4
004DC1C3  .~75 1E           JNZ SHORT DVDMenuS.004DC1E3
004DC1C5  . 50              PUSH EAX
004DC1C6  . 8B47 1C         MOV EAX,DWORD PTR DS:[EDI+1C]
004DC1C9  . 50              PUSH EAX
004DC1CA  . FF15 B4575E00   CALL DWORD PTR DS:[<&USER32.KillTimer>]
004DC1D0  . 6A 00           PUSH 0
004DC1D2  . 6A 00           PUSH 0
004DC1D4  . 68 FC656400     PUSH DVDMenuS.006465FC
004DC1D9  . E8 6E9D0D00     CALL DVDMenuS.005B5F4C
004DC1DE  .~E9 94000000     JMP DVDMenuS.004DC277
```

```
004DBD90  . 50              PUSH EAX
004DBD91  . 64:8925 00000(  MOV DWORD PTR FS:[0],ESP
004DBD98  . 81EC 68010000   SUB ESP,168
004DBD9E  . 8B45 08         MOV EAX,DWORD PTR SS:[EBP+8]
004DBDA1  . 53              PUSH EBX
004DBDA2  . 56              PUSH ESI
004DBDA3  . 57              PUSH EDI
004DBDA4  . 83F8 01         CMP EAX,1
004DBDA7  . 8BF9            MOV EDI,ECX
004DBDA9  .~75 53           JNZ SHORT DVDMenuS.004DBDFE
004DBDAB  . 50              PUSH EAX
004DBDAC  . 8B47 1C         MOV EAX,DWORD PTR DS:[EDI+1C]
004DBDAF  . 50              PUSH EAX
```

```
004DBE4D  .~0F85 54010000   JNZ DV
004DBE53  . 6A 00           PUSH 0
Stack SS:[0019EE70]=00000003
EAX=0019EED4
```

결국에 메시지를 출력하기 전에 EAX, 4 와 비교하고, 같아야지만 성공 메시지가 출력된다. 즉, EAX는 4여야 한다.

다시 코드를 살펴보면 EBP+8부분에는 3이 들어있다. 해당 주소에 값을 변경하고, 패치하려 했지만, 그 순간에만 적용되고, 다시 실행하면, 똑같이 3이 들어온다. 왜냐하면 ASLR 보호기법이 적용돼 매 실행마다 주소가 바뀌기 때문이다. 그래서 패치를 해줘야된다. MOV EAX, 4를 해주면 되는데, 이렇게 할 경우 명령 크기가 5바이트가 돼 주소값 자체가 바뀌게 된다. 왜냐하면 기존 명령이 3바이트이기 때문이다. 이 때 필요한게 인라인 패치다.



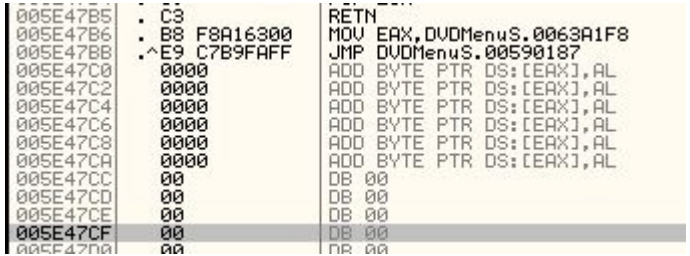
궁금해서 MOV EAX, 4로 패치해봤다. 그런데 등록 성공 메시지가 무한으로 나왔다. 왜 무한으로 바졌는지 해당 프로그램을 분석해봤다. 패치한 주소는 이전과 다를거 없이 바뀐게 없다. 하지만 하나씩 트레이싱 해봤는데, 주소만 안바뀌었지 명령 실행하는 우선순위가 바뀌어 하나씩 땡겨져서 실행되는것 같다. CALL해야하는 부분에서 계속 jmp가 되면서 해당 명령과 다른 동작을 진행했다.

다음과 같이 해당 CALL 명령 다음에 JMP가 동작하는데, 해당 명령에서 jmp가 진행돼 원래 있던 곳으로 갔다.

그래서 3바이트 명령으로 바꿔줘야한다. 그중 한가지 방법이 떠올랐다. MOV AX, 4를 쓰면 3바이트가 되고, 인라인 패치를 굳이 안해도 정상적으로 패치되었다.

이번에는 또 다른 인라인 패치를 통해 코드를 패치해볼 것이다.

해당 영역을 MOV EAX, 4로 패치하자니 5바이트가 돼서 그 뒤 명령을 덮어씌울 것이다. 그래서 다른 공간에 해당 부분을 복사하고, 그 영역으로 분기해서 수행한 뒤 다시 돌아오는 방법을 쓸 것이다.

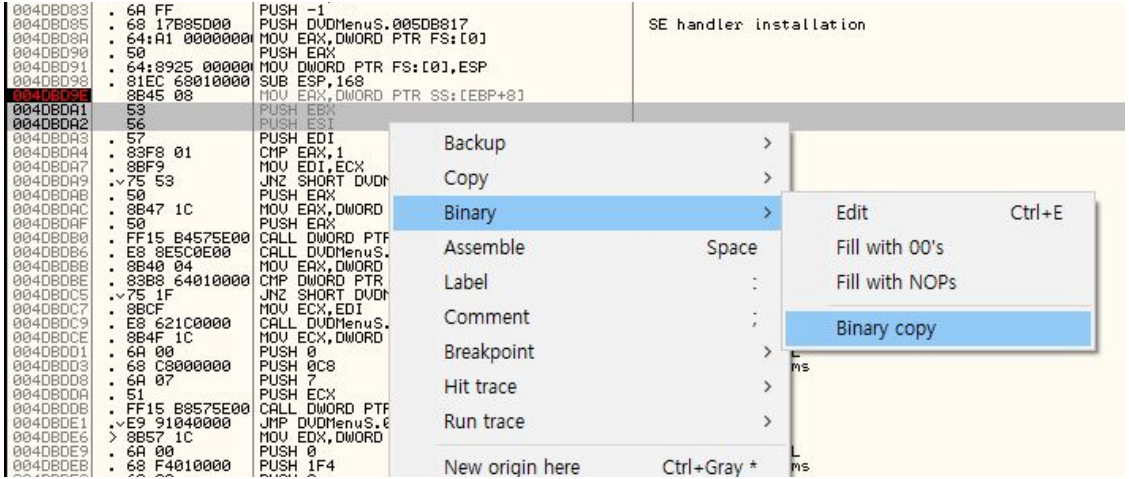


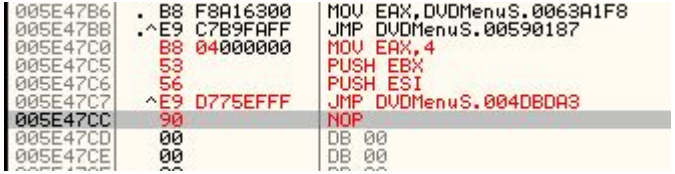
코드 끝으로 가면 잉여 공간이 있다.
(PE 구조는 파일 또는 메모리에서
섹션의 시작위치는 각각 최소 기본
단위의 배수에 해당하는 위치여야
하므로 빈공간은 NULL로
채워버리기 때문에 잉여 공간이
있다고한다. 이 부분을 이용해
인라인 패치를 할
것이다.(Alignment))

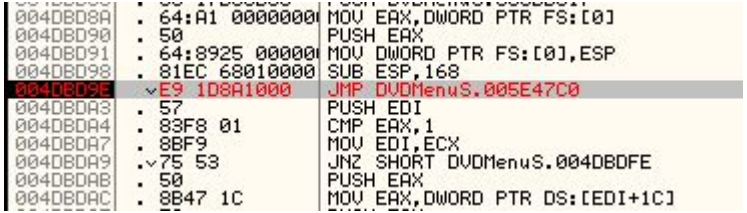
이 영역으로 분기할 것이다.
분기하기 위해 long 점프를
해야한다.

어셈 코드에서 점프를 하고자 할 때 점프하고자 하는 주소를 기준으로 점프 종류가 2가지로 나뉜다.

short 점프는 주소 공간을 2 바이트만 차지하지만, 점프 명령어가 실행되는 주소와, 점프 대상 주소를 기준으로 80바이트 내로 밖에 점프하지 못한다. 그 이상 점프하고 싶으면 long 점프를 사용해야 한다. 하지만 이는 주소 공간을 5바이트 차지하기 때문에 잉여 공간으로 이동해 명령을 수행한 다음 다시 돌아와야한다.







해당 명령을 복사한 후 잉여 공간에 복사한다. 그리고 원래 명령을 JMP [잉여 공간 주소]로 패치한다.

패치한 후 실행하니 무한루프에 빠지게 됐다. 실행을 하면 해당 부분으로 다시 돌아오게 된다.

원인을 분석하기 위해 하나씩 다시 분석해본다.