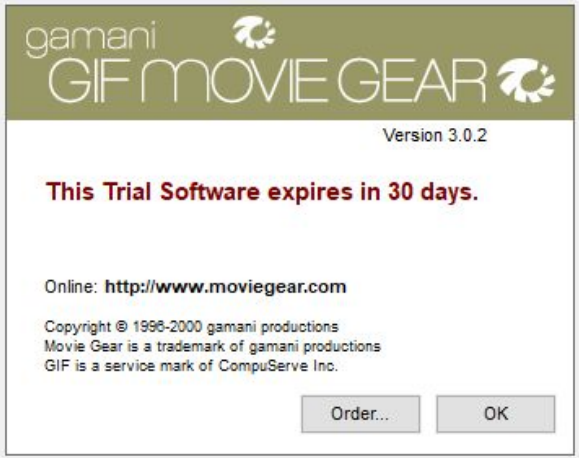
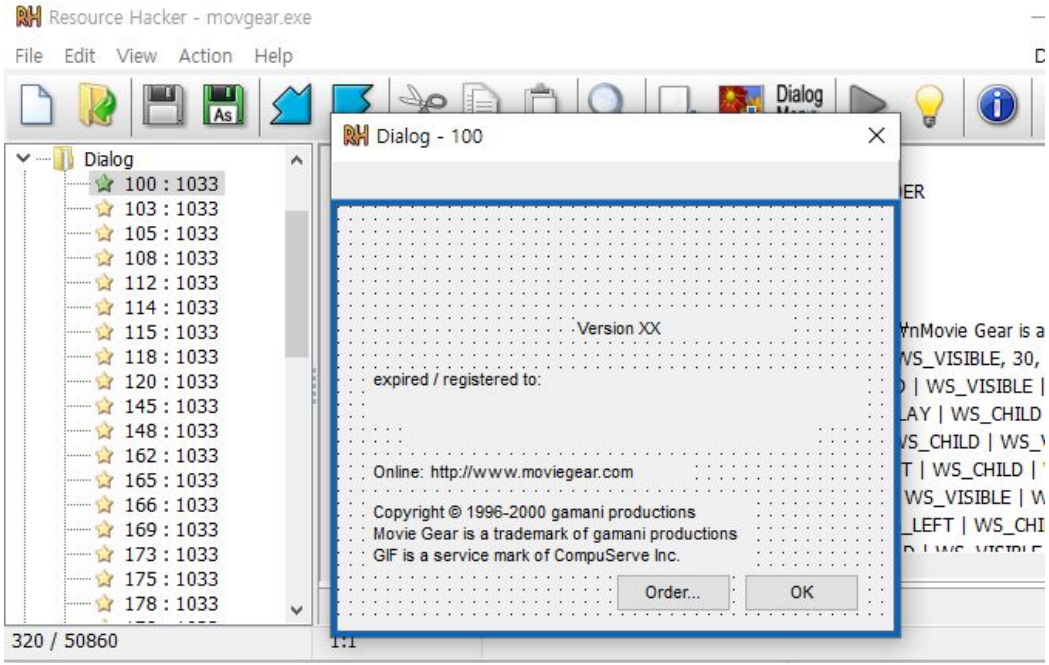
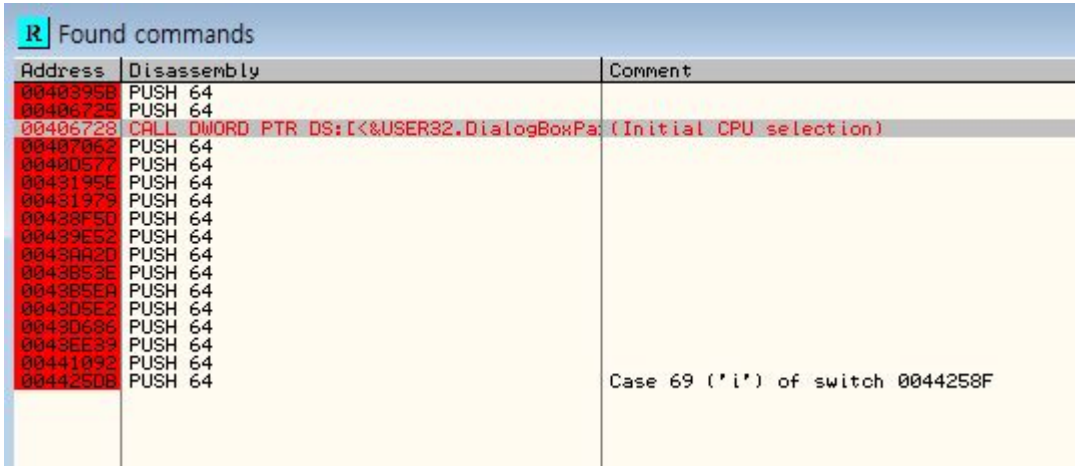


프로그램을 실행하고, 종료할 시 해당 Nag창이 뜬다. 이 Nag창을 Resource Hacker 툴을 써서 없애는게 이번 리버싱의 목적이다.

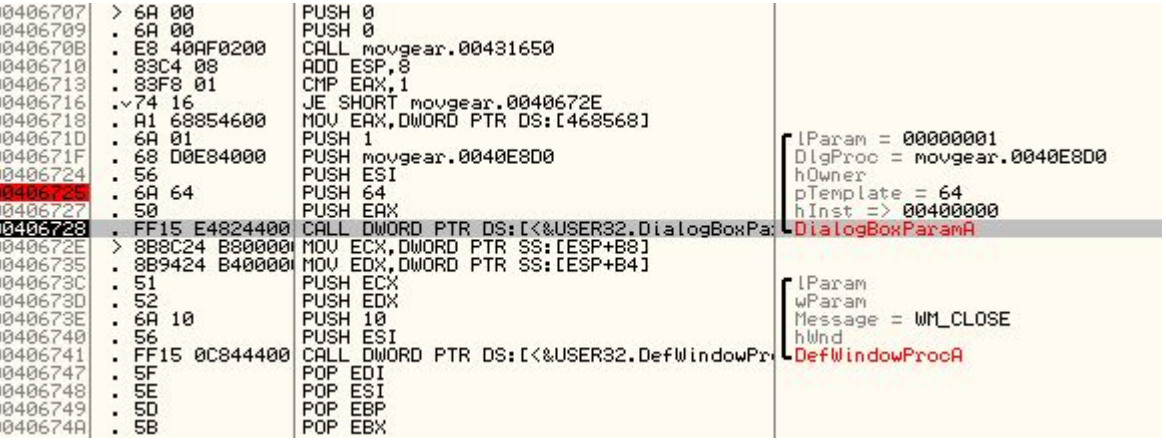


Resource hacker로 살펴보면 100번대에 아까 봤던 Nag창을 볼 수 있다. 100을 헥스 값으로 표현하면 64다.

push 64로 해당 명령을 사용하는 부분을 검색해본다.



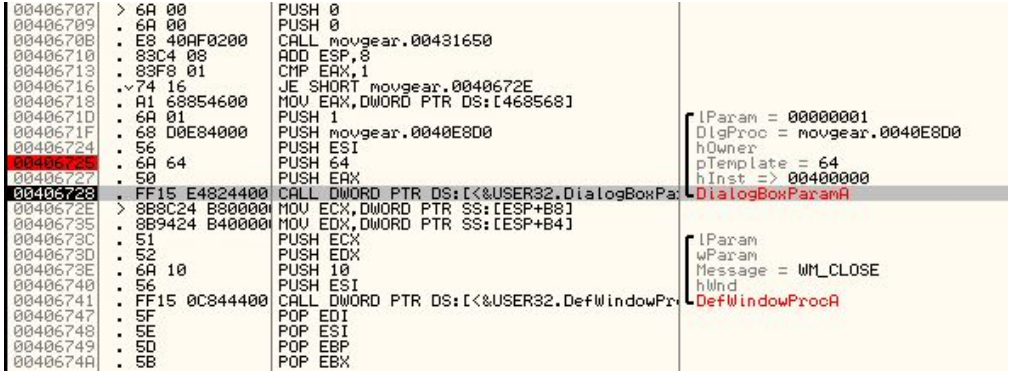
push 64를 검색해보면 다양한 부분에서 사용하는걸 알 수 있다. 모든 부분에 bp를 걸고 하나씩 트레이싱 해본다.



프로그램이 실행되고, 프로그램을 종료시키면 해당 부분에서 bp가 걸린다. 아마 이 부분이 Nag 창을 호출하는 부분인것같다. push 64중에 64가 무엇인지 찾아보니까 식별자라고 한다. 즉, DialogBoxParam을 사용하는 함수를 하나의 정수값으로 표현해 구별한다는 의미이다.

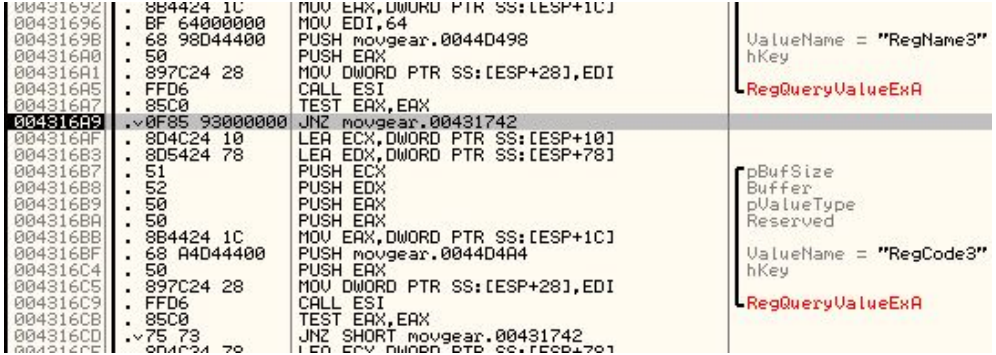
위쪽에 보면 분기문이 있다 조건에 만족하지 못하면 분기한다.

Nag창은 사라졌지만, 등록은 아직 안된 상태이다. 이걸 우회에 등록해보도록 하겠다.



위 분기문 위에 보면 함수가 있다. 이 함수에 의해 EAX가 결정되고 같을 경우 분기된다. 해당 함수를 분석해본다.

함수를 보니 RegOpenKeyExA를 통해 프로그램을 레지스터에 등록하고 있다. 이 함수에서 프로그램 등록 여부를 체크하는 것 같다.



아까 EAX에 따라 분기 여부를 결정한다고 했다. 프로그램 등록 관련 함수 RETN이 실행되기 전 MOV EAX, EBX를 통해 EAX 값을 결정하고 있다.

함수를 처음부터 실행시켜 보면 RegQueryValueExA를 통해 레지스터 키를 비교하고 있다. 두번의 API가 호출되는데, 해당 부분은 Name과 시리얼 번호가 있는지 없는지 확인하는 것이다. 등록된 시리얼 키가 없으니 조건문을 만족하지 못하고, 431742함수로 분기될것이다.

분기하니 함수의 끝부분으로 이동하고, 이대로 MOV EAX, EBX를 통해 진행하면 아까와 같은 오류를 만날것이다. 그렇기에 해당 명령을 패치해줘야한다. 지금 EBX값은 FFFFFFFF이고, 이 값을 옮기면 안되기 때문에 MOV EAX, EBX -> MOV AL, 1로 바꿔줄 것이다. 이유는 원래 명령이 2바이트로 되어있기 때문에 똑같이 2바이트 크기로 패치 해줘야하기 때문이다. 그렇지 않으면 뒤에 명령을 덮어씌울것이다.

```
. F3:A4          REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:
> 8B4C24 0C      MOV ECX,DWORD PTR SS:[ESP+C]
. 51             PUSH ECX                                 hKey
. FF15 00804400  CALL DWORD PTR DS:[<&ADVAPI32.RegCloseK  RegCloseKey
. 5F             POP EDI
  B0 01          MOV AL,1
. 5E             POP ESI
. 5B             POP EBX
. 81C4 D0000000  ADD ESP,0D0
.. C3            RETN
```

이렇게 패치를 해주면 된다. 이러면 아까 Nag창을 없애는 패치를 굳이 하지 않아도 Nag창이 뜨지 않는다.