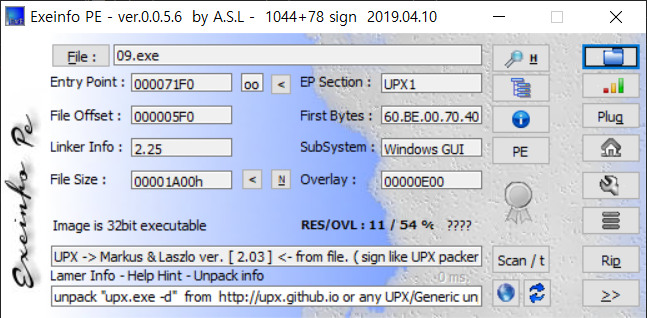
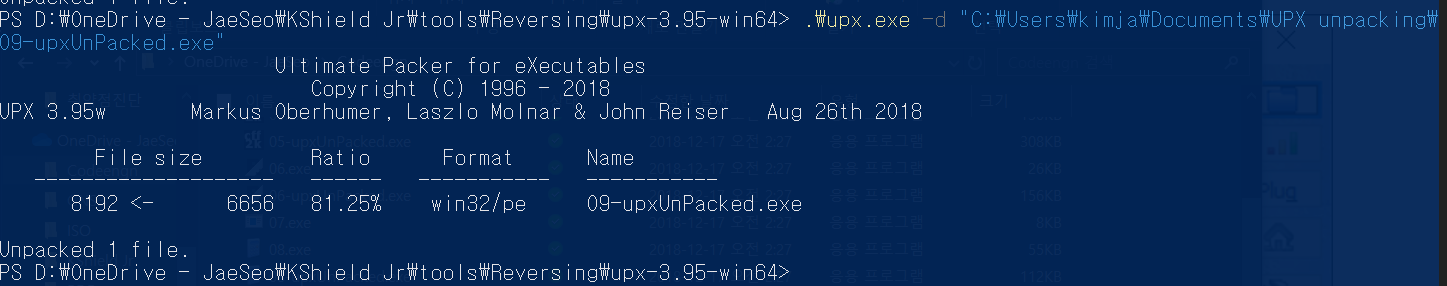
**09.exe -** StolenByte를 구하시오 Ex) 75156A0068352040

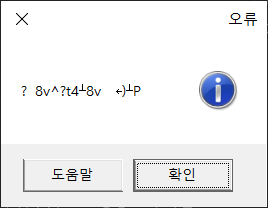
일단 PE 분석을 해본다.



그 결과 UPX로 Packing 되어 있는 것을 볼 수 있다. UPX로 UnPacking 한다.



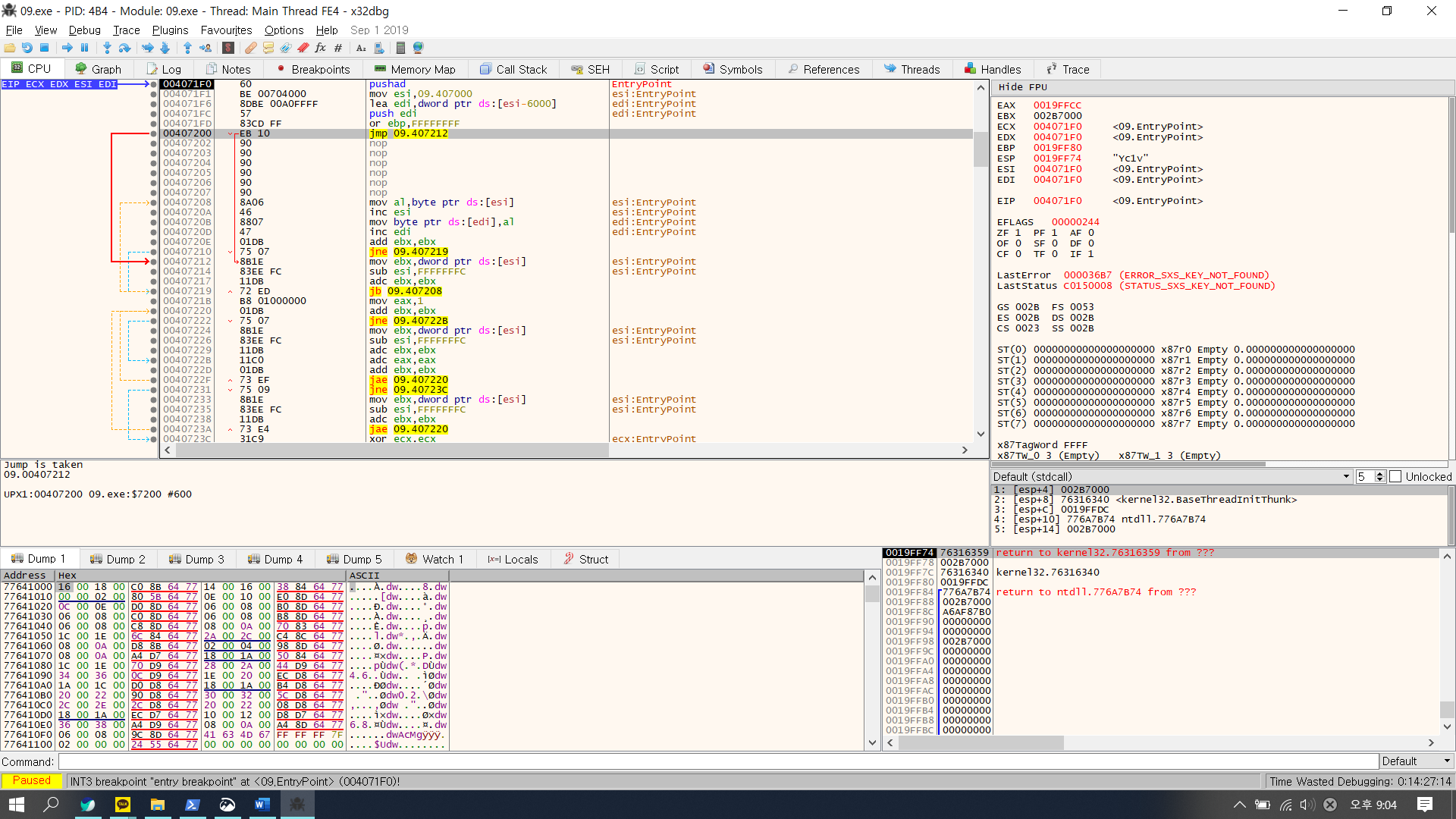
이제 프로그램을 실행하여 분석을 한다.



Upx로 Unpacking를 하고 나니 문자열이 다 깨지고 오류 메세지를 출력하고 있다.

Upx에서 StolenByte를 일부 가지고 있어서 오류가 나는 것으로 예상이 된다.

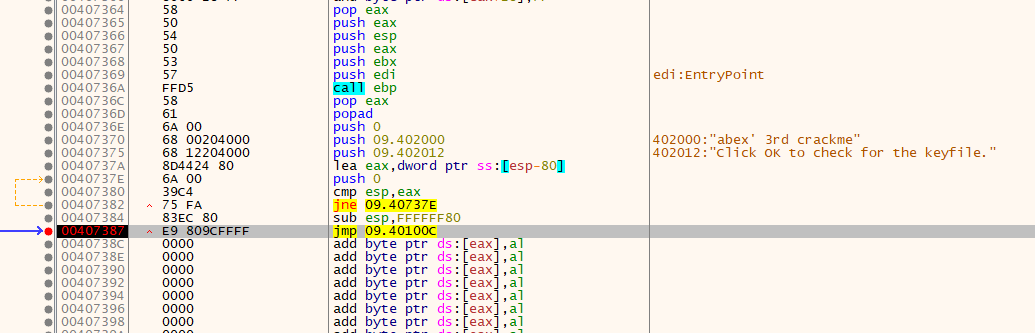
그래서 이번에는 upx Unpacking을 진행하지 않고 디버깅을 진행하도록 한다.



일단 x32dbg로 분석을 한다. 실행을 해보면 처음 upx로 unpacking을 진행하는 모습을 볼 수 있다.



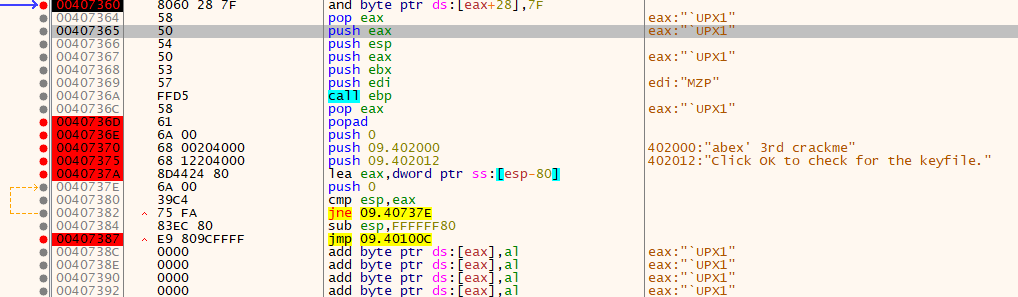
Upx로 unpacking을 하면서 계속 반복을 하는 모습을 볼 수 있는데 F8과 Break Point 그리고 F9를 활용하여 반복문에 걸리면 빠져나오도록 하여 분석을 진행한다.



진행을 계속 하다 보면 jmp을 통해 ep로 이동하는 모습을 볼 수 있다. 이제 여기에 Break Point를 걸고 분석을 시작한다.

이때 앞에 정상적으로 작동을 하였다면 보이는 메시지가 보인다.

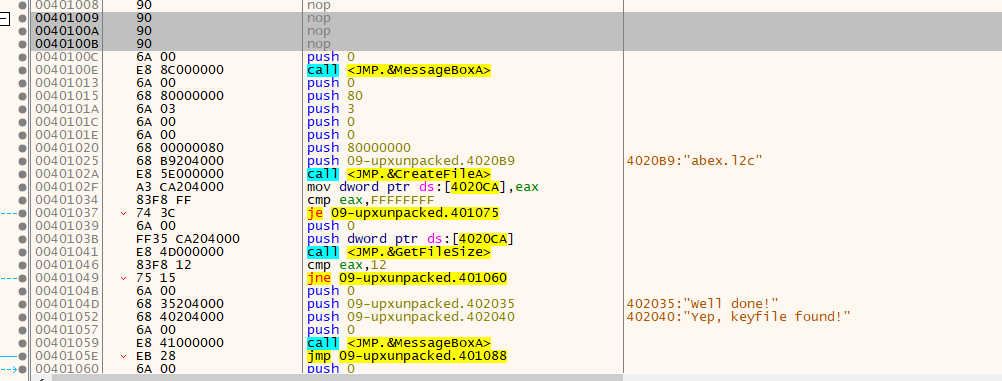
StolenByte로 추정이 된다 앞에 BreakPoint를 걸고 분석을 해본다.

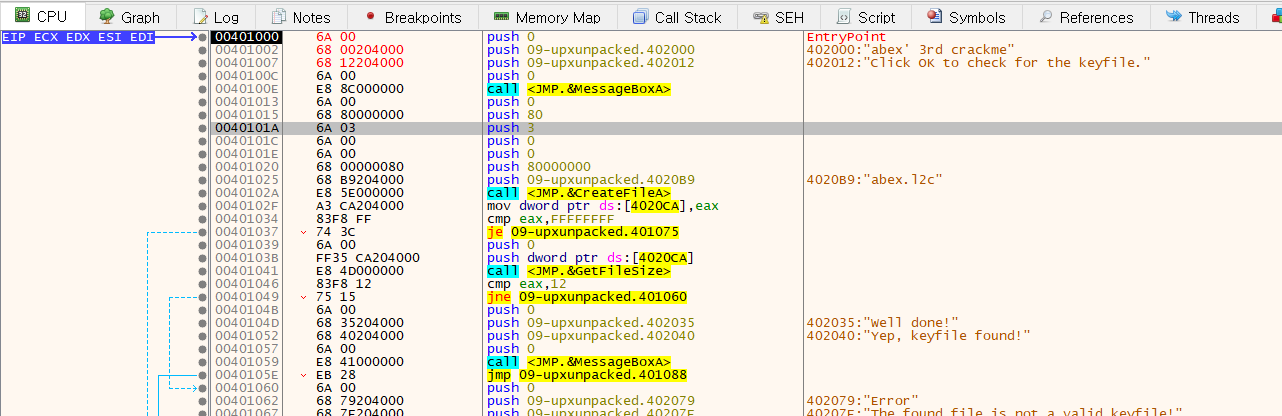


Popad을 통해 oep에서 작동할 레지스터를 복구를 한 다음 StolenByte로 추정이 되는 명령들을 실행한다.

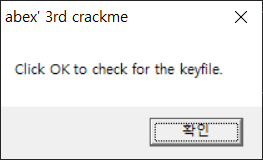
이때 push로 스택에 미리 명령어를 삽입 후 반복적으로 0을 집어넣어 esp와 eax를 같은 값으로 바꾸고 JMP를 하는 것을 볼 수 있다.

이제 실제로 이 코드를 UnPacking 된 파일에 집어넣어서 확인을 해본다.





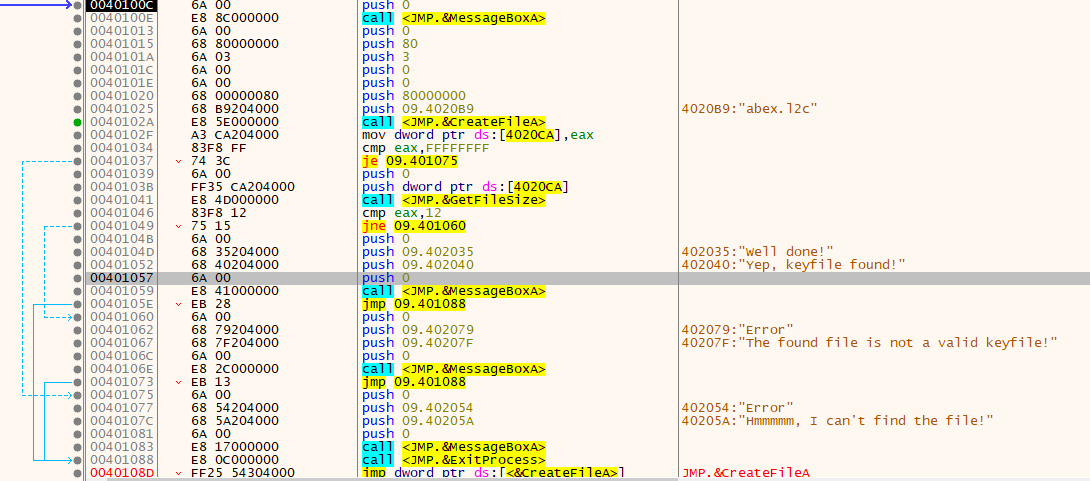
NOP으로 채워져 있던 공간과 정확하게 일치를 하는 것을 볼 수가 있다. 실제로 실행을 시켜서 정상적인지 확인해본다.



위와 같은 모습으로 정상적으로 작동하는 모습을 볼 수 있다.



**정답: 6A0068002040006812204000**



Main 함수 내부로 추정이 되는 부분이다.

일단 처음에는 안내 메시지를 출력한 다음에 “CreateFileA”라는 함수를 호출 하여 파일이 잇는지 없는지 체크를 한 다음 만약 있다면 “GetFileSize” 함수를 통해 키파일이 맞는지 에 대해 확인을 하고 메시지를 출력 하는 것으로 예상이 된다.

이제 직접 실행을 하여 어떻게 진행이 되는지 분석해본다.