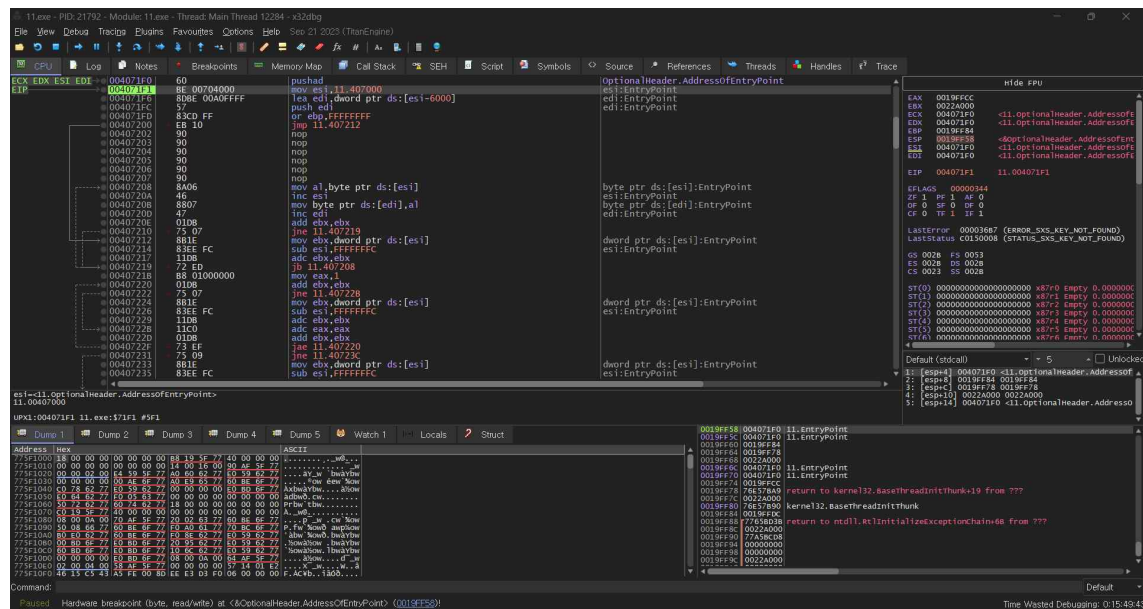


CodeEngn Basic RCE

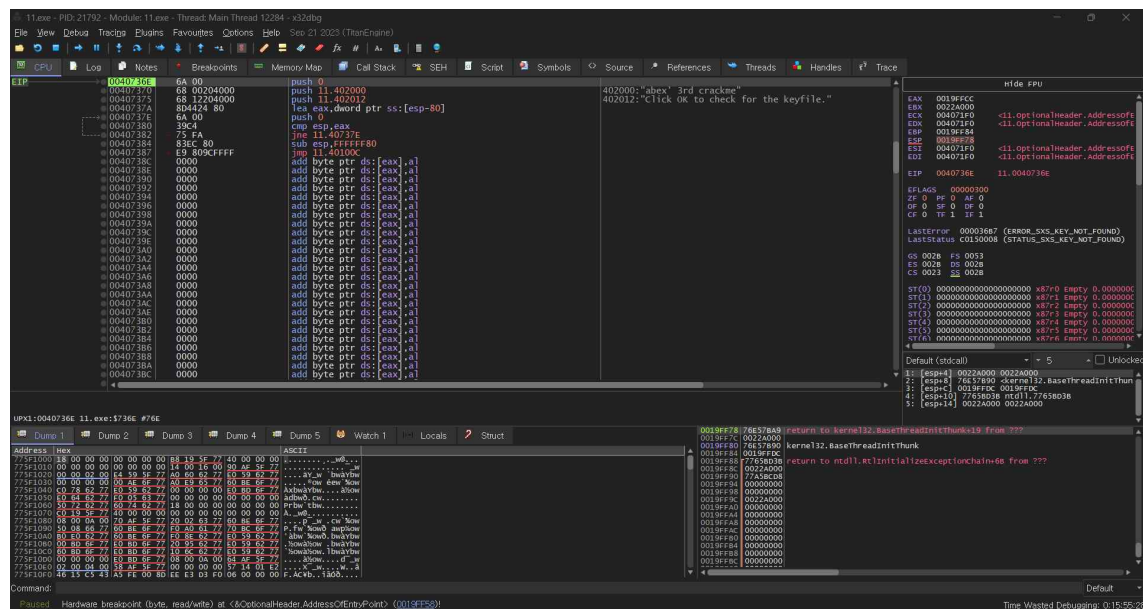
강 보 성

[11]

풀이 : pushad 명령어를 실행하고 ESP에 저장된 주소로 이동하여 popad 명령어 실행 이후의 위치를 찾는다. pushad ~ popad 사이의 과정은 패킹된 코드를 언패킹(원본 코드를 메모리상에 복구)하여 원본 코드를 받아오는 과정이다. pushad를 통해 레지스터의 값을 보존하고, 언패킹 후 popad를 통해 이전의 레지스터 값을 복원하여 원본 코드를 사용할 수 있게 한다.

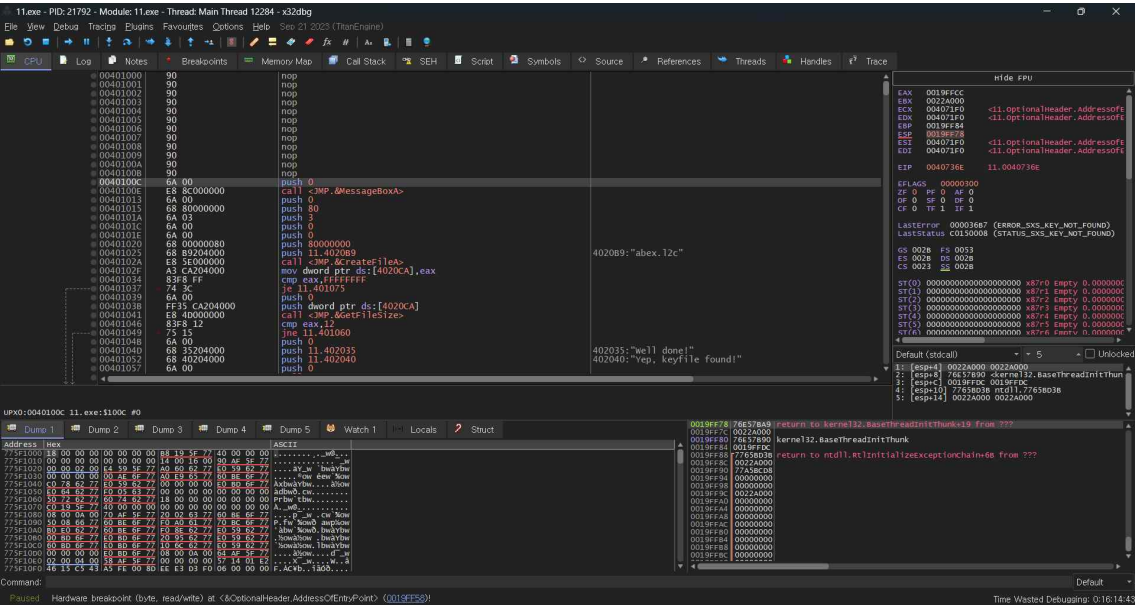


<pushad 명령어 실행 후 ESP에 레지스터 값이 저장된 모습>



<popad 이후 코드>

해당 위치에서 StolenByte로 의심되는 문자열 데이터를 발견할 수 있다.
또 아래에 원본 코드로 점프하는 코드가 보인다. 해당 코드의 주소를 따라가 보면 원본 코드를 발견할 수 있다.



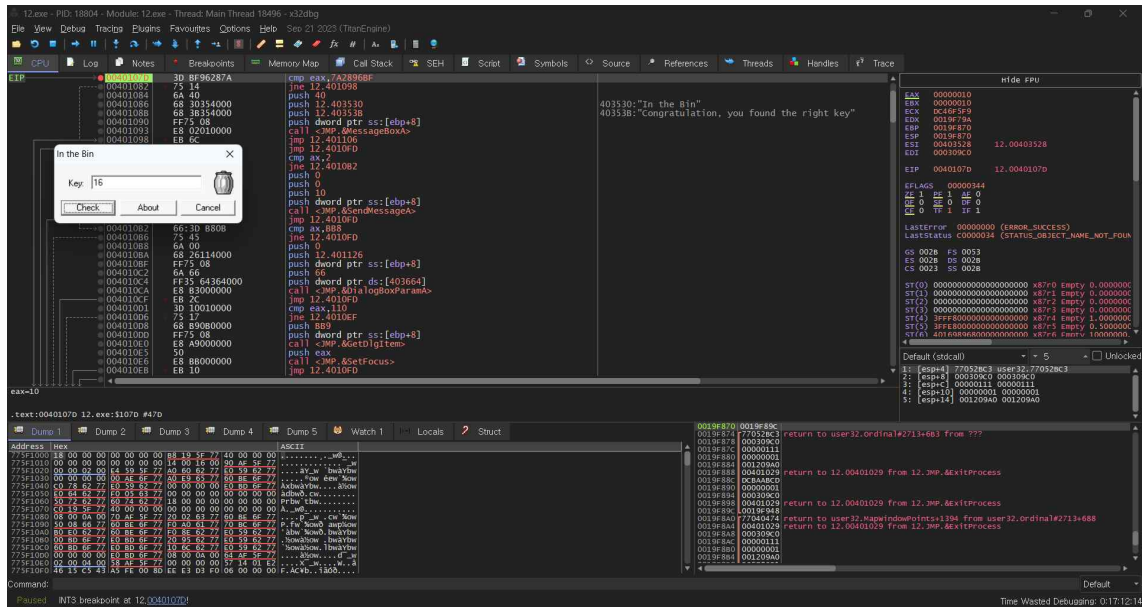
<원본 코드>

원본 코드를 보면 위쪽으로 의미 없는 값이 채워져 있음을 확인할 수 있다. 이것으로 원본 코드에 StolenByte 기법이 적용되어 있음을 알 수 있다. 동시에 원본 코드의 OEP가 00401000임을 알 수 있다.

답 : 00401000 6A0068002040006812204000

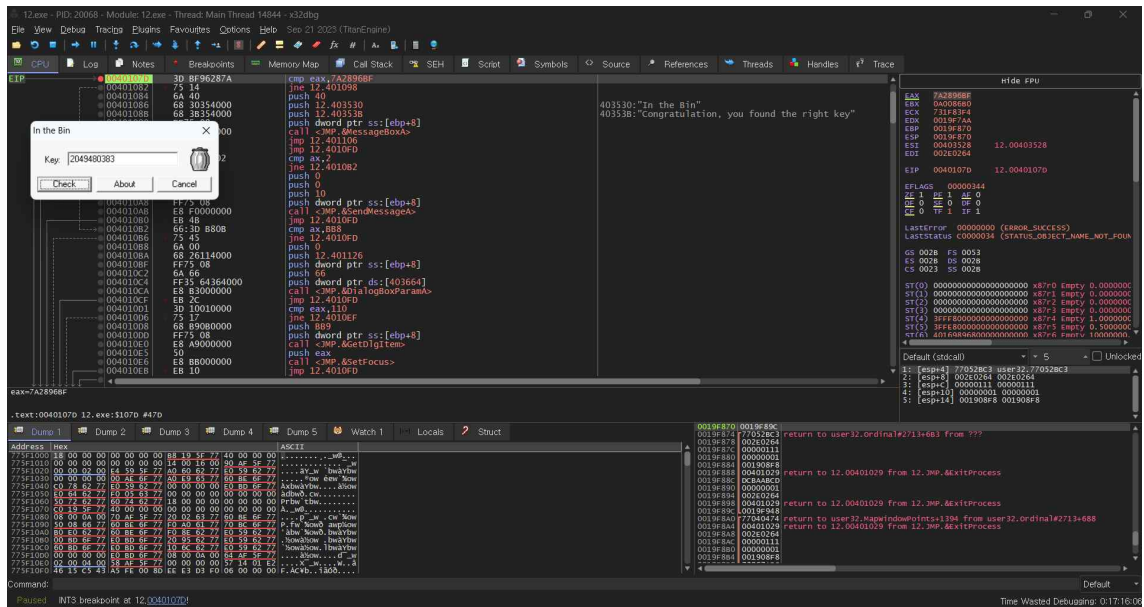
[12]

풀이 : 문자열 데이터를 분석하다 보면 성공 구문이 보인다. 그 위로 cmp 명령어가 보이는데 eax 값과 7A2896BF라는 값을 비교하여 같으면 성공구문을 출력하는 것으로 보인다. 실제로 cmp 명령어에 브레이크 포인트를 걸어두고 디버깅을 하면 입력값이 16진수로 변환되어 eax에 저장되는 것을 확인할 수 있다.



<16을 입력한 뒤 변화된 eax 값>

따라서 프로그램을 다시 실행하여 7A2896BF라는 값을 10진수로 변환한 값인 2049480383을 입력하면 성공 구문이 출력됨을 확인할 수 있다.



<2049480383을 입력한 뒤 변화된 eax 값>

다음으로, Key 값을 성공메시지 대신 출력하기 위해서 헥스에디터를 사용하여 성공메시지 문자열을 2049480383으로 overwrite한다.

12.exe

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded text

00000C30 4B 48 31 78 4D 2F 4C 72 59 58 37 33 36 67 4E 2B kHixd/LrTX736gN+

00000C40 43 43 4D 61 78 36 49 65 62 37 31 37 6C 4D 4E 6A CCMa6Ieb7171HhJ

00000C50 4E 34 35 47 65 53 51 48 35 4A 78 70 38 72 67 7A H49GeSQ85Jxp8rgz

00000C60 44 4B 39 34 66 34 36 34 4D 71 77 31 4D 69 4E 75 dR94f46mqvIMhnu

00000C70 59 52 6A 39 6B 6C 4C 4F 4C 61 47 6D 75 6C 2F 4B YRj9k110LaGm1/R

00000C80 6C 71 66 64 5A 2F 31 72 62 4D 75 38 66 62 69 58 lqfdG/1rbMu5fhhX

00000C90 46 50 73 6F 47 33 78 70 4E 34 6C 77 5A 33 42 FPa0GkygM4lwZ3B

00000CA0 30 32 4D 70 62 44 37 4D 78 69 33 63 4E 34 78 02MqbD7hx13cHntxk

00000CB0 5A 39 73 62 35 59 4B 6D 4D 42 6C 36 68 47 65 42 Z9ab5XKmB1chGeB

00000CC0 35 67 41 41 47 56 62 6D 68 35 42 44 52 4C 59 6B SgAGVtmh5BDR1Kx

00000CD0 61 47 78 16 45 49 55 39 76 52 78 6C 32 56 64 62 a9aFeH09wRk12Vdb

00000CE0 51 59 33 59 62 58 74 57 47 34 6E 67 61 72 49 72 QY3hXtW64ngapIz

00000CF0 6D 65 33 67 65 6A 6D 62 42 66 4E 4C 4C 2F 6A 57 me3gejmb8fHLL/JW

00000D00 50 63 30 4A 49 59 34 62 47 2B 47 45 51 77 4C 72 PcoJIT4hg+GEQwLz

00000D10 36 4B 70 47 6C 7A 51 66 49 53 4D 6A 4D 2F 34 6A kxp01aQFISNM/4j

00000D20 62 34 45 68 4F 71 69 71 00 00 00 00 78 56 34 12 b4EhOqig....xV4.

00000D30 49 6E 20 74 68 65 20 42 69 6E 00 H3 6F 6E 67 7A In the Bin..cong

00000D40 61 74 75 6C 61 74 69 6F 6E 2C 20 79 6F 75 20 64 atulation, you f

00000D50 6F 75 6E 6A 20 78 6B 65 20 72 69 67 68 74 20 68 80383.....

00000D60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000D70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000D80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000D90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000DA0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000DB0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000DC0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000DD0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000DE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000DF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000EA0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000EB0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

오브셋(h): D38 블록(h): D38-D61 길이(h): 27 메모리 쓰기

<성공메시지 영역 선택>

12.exe

Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F Decoded text

00000C30 4B 48 31 78 4D 2F 4C 72 59 58 37 33 36 67 4E 2B kHixd/LrTX736gN+

00000C40 43 43 4D 61 78 36 49 65 62 37 31 37 6C 4D 4E 6A CCMa6Ieb7171HhJ

00000C50 4E 34 35 47 65 53 51 48 35 4A 78 70 38 72 67 7A H49GeSQ85Jxp8rgz

00000C60 44 4B 39 34 66 34 36 34 4D 71 77 31 4D 69 4E 75 dR94f46mqvIMhnu

00000C70 59 52 6A 39 6B 6C 4C 4F 4C 61 47 6D 75 6C 2F 4B YRj9k110LaGm1/R

00000C80 6C 71 66 64 5A 2F 31 72 62 4D 75 38 66 62 69 58 lqfdG/1rbMu5fhhX

00000C90 46 50 73 6F 47 33 78 70 4E 34 6C 77 5A 33 42 FPa0GkygM4lwZ3B

00000CA0 30 32 4D 70 62 44 37 4D 78 69 33 63 4E 34 78 02MqbD7hx13cHntxk

00000CB0 5A 39 73 62 35 59 4B 6D 4D 42 6C 36 68 47 65 42 Z9ab5XKmB1chGeB

00000CC0 35 67 41 41 47 56 62 6D 68 35 42 44 52 4C 59 6B SgAGVtmh5BDR1Kx

00000CD0 61 47 78 16 45 49 55 39 76 52 78 6C 32 56 64 62 a9aFeH09wRk12Vdb

00000CE0 51 59 33 59 62 58 74 57 47 34 6E 67 61 72 49 72 QY3hXtW64ngapIz

00000CF0 6D 65 33 67 65 6A 6D 62 42 66 4E 4C 4C 2F 6A 57 me3gejmb8fHLL/JW

00000D00 50 63 30 4A 49 59 34 62 47 2B 47 45 51 77 4C 72 PcoJIT4hg+GEQwLz

00000D10 36 4B 70 47 6C 7A 51 66 49 53 4D 6A 4D 2F 34 6A kxp01aQFISNM/4j

00000D20 62 34 45 68 4F 71 69 71 00 00 00 00 78 56 34 12 b4EhOqig....xV4.

00000D30 49 6E 20 74 68 65 20 42 69 6E 00 32 30 34 39 34 In the Bin..20494

00000D40 38 30 33 38 33 00 00 00 00 00 00 00 00 00 00 00 80383.....

00000D50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000D60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000D70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000D80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000D90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000DA0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000DB0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000DC0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000DD0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000DE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000DF0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000E80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

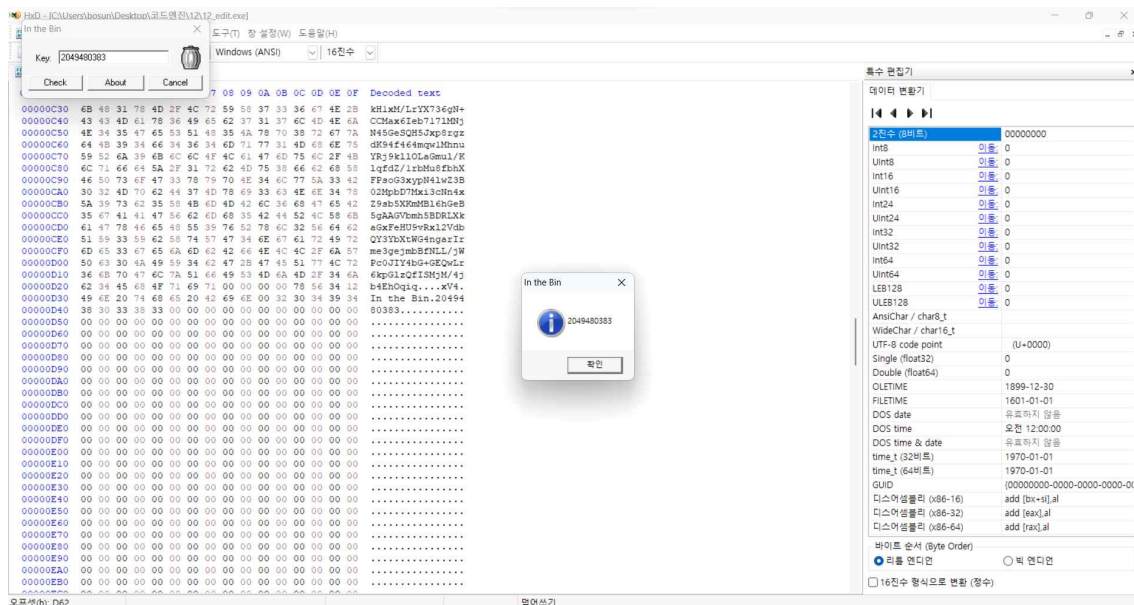
00000E90 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000EA0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00000EB0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

오브셋(h): D62 * 변경됨 * 메모리 쓰기

<overwrite(0D3B~0D45)>

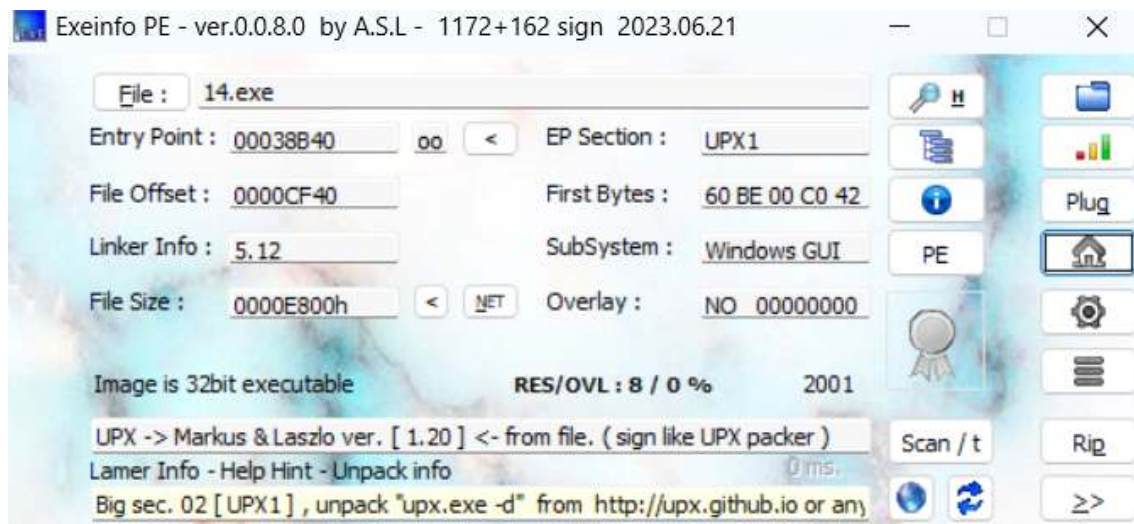


<성공메시지 대신 Key 값 출력>

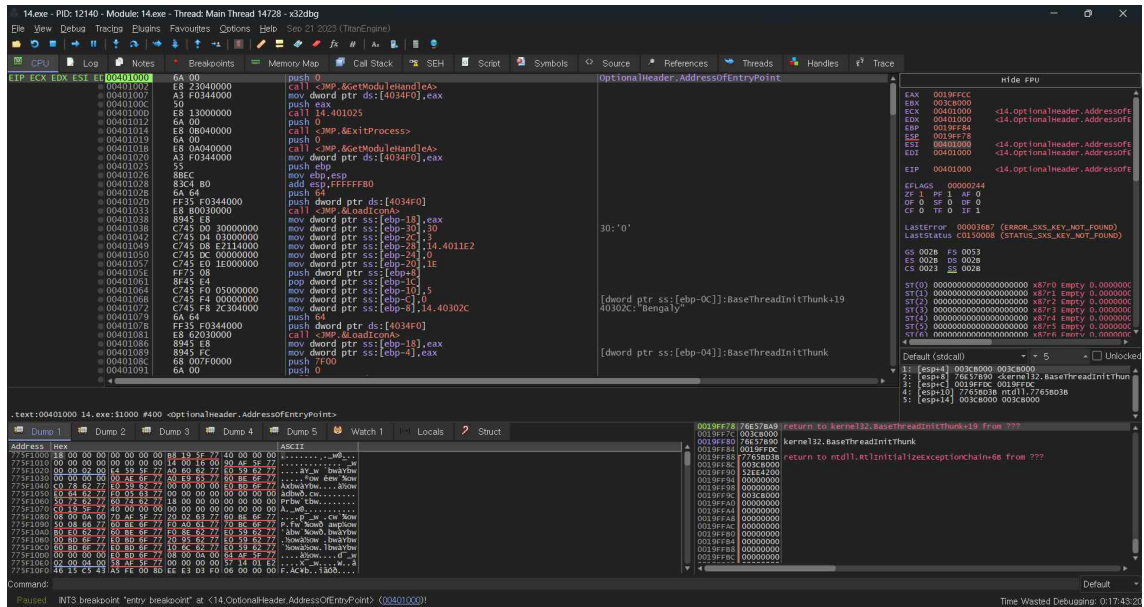
답 : 2049480383 0D3B 0D45

[14]

풀이 : 14.exe가 upx로 패킹되어있기 때문에 원활한 분석을 위해 언패킹한다.

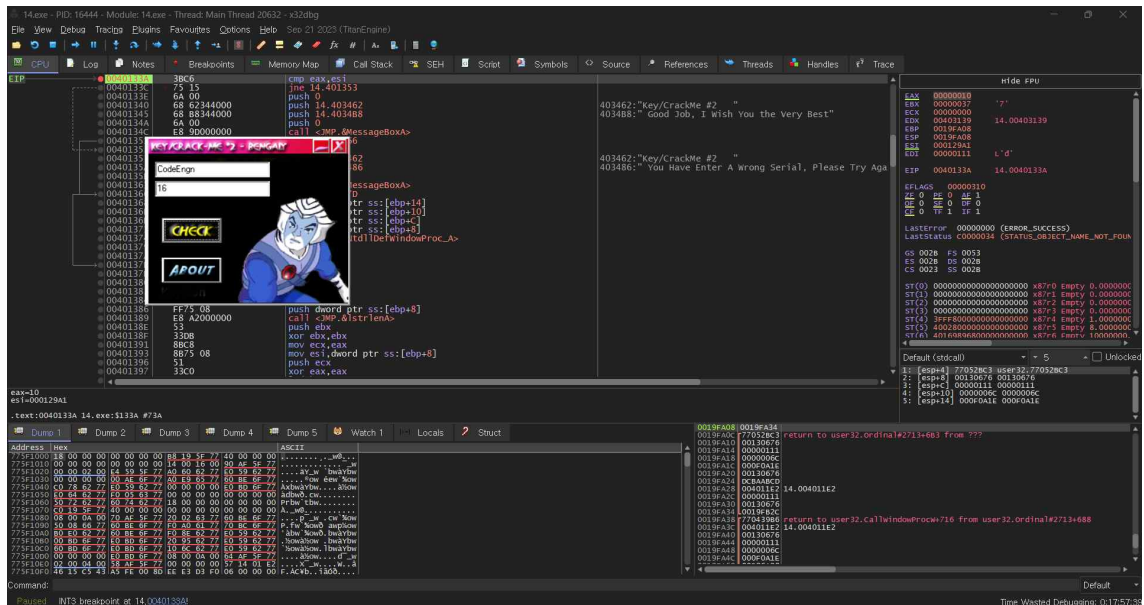


<Exeinfo PE에서 확인한 14.exe 정보>



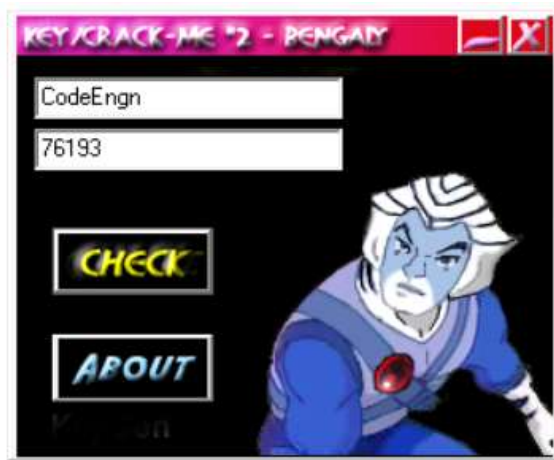
<언패킹 후 14.exe 디버깅 화면>

그 후 문자열 데이터를 분석하다 보면 성공 메시지를 발견할 수 있는데, 성공 메시지 위로 eax와 esi를 비교하는 cmp 명령어가 보인다. 해당 명령어에 브레이크 포인트를 걸어두고 다시 실행하여 시리얼 값으로 16을 입력한다. 그러면 eax에 10이 저장됨과 esi에 129A1가 저장됨을 확인할 수 있다.



<시리얼 값으로 16을 입력 후, 변환된 eax 값과 esi 값>

내가 입력한 값을 16진수로 변환시키고 129A1과 비교한 뒤 같으면 성공 메시지를 출력하는 것으로 추측된다. 따라서 프로그램을 다시 실행시키고 129A1을 10진수로 변환시킨 값인 76193을 시리얼값으로 입력해본다. 그러면 성공 메시지가 출력됨을 확인할 수 있다.

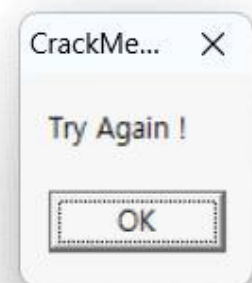
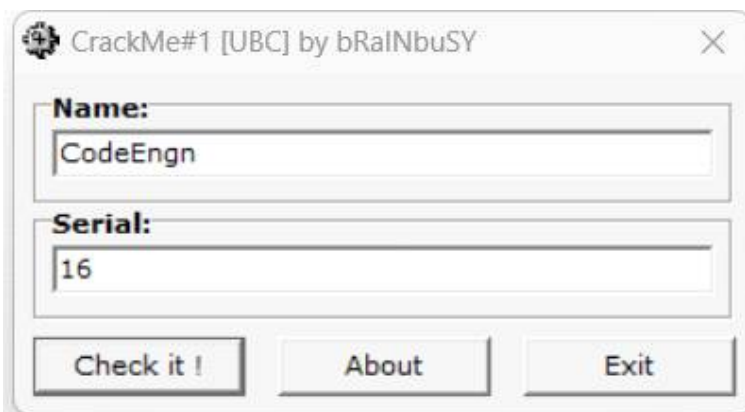


<성공 메시지 출력>

답 : 76193

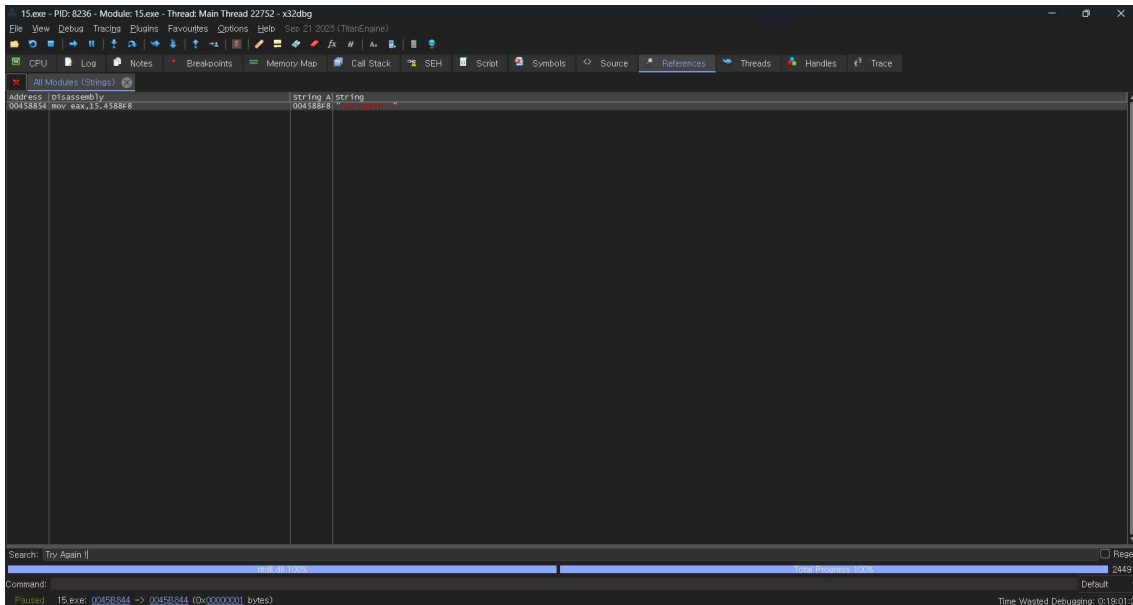
[15]

풀이 : 디버깅을 하기에 앞서 프로그램을 실행하여 임의의 값을 시리얼 값으로 입력해본다. 그러면 Try Again ! 이라는 메시지를 확인할 수 있다.



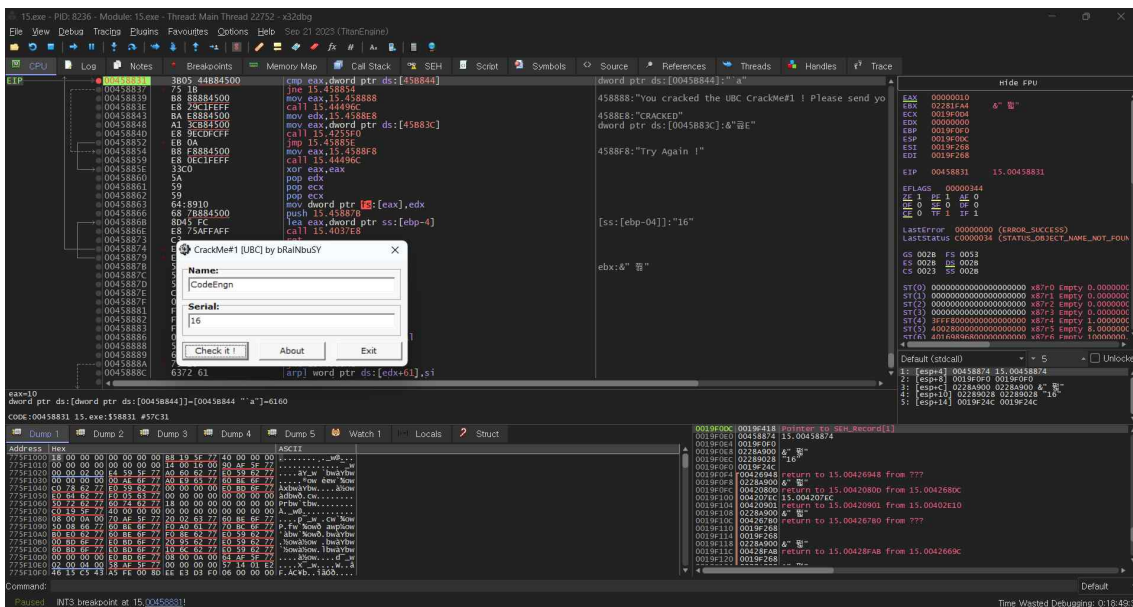
<프로그램 실행>

15.exe를 x64dbg로 열어서 Try Again ! 이라는 문자열을 검색하여 해당 주소로 이동한다. 그러면 성공 메시지를 확인할 수 있다.



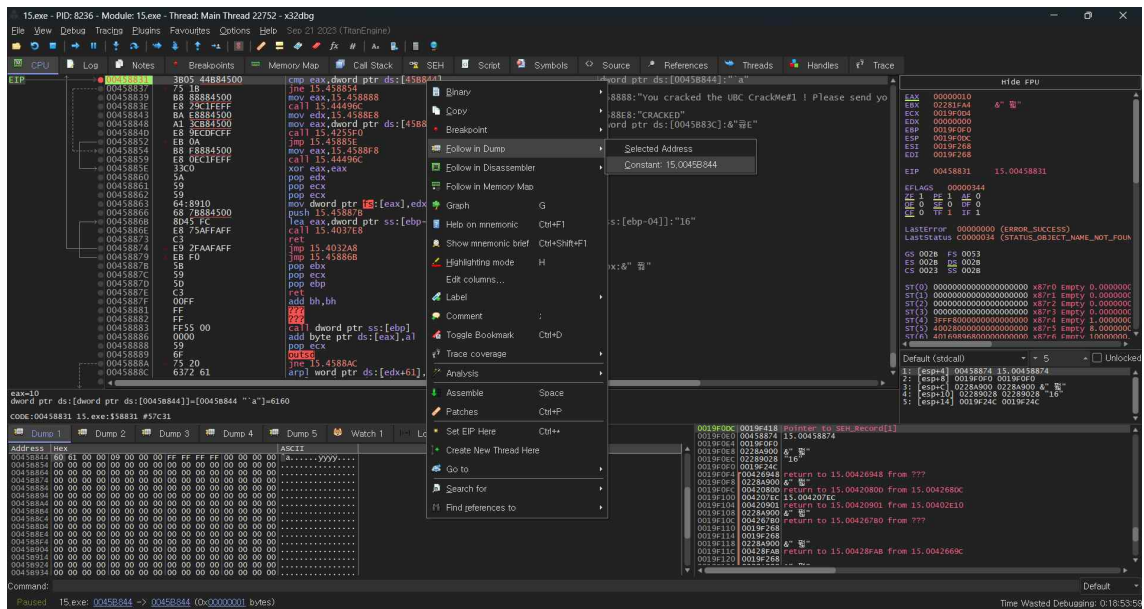
<“Try Again !” 문자열 검색>

또 성공 메시지 위 쪽으로 `eax`와 `ds:[45B844]` 데이터를 비교하는 `cmp` 명령어를 발견할 수 있다. (*ds란 세그먼트 레지스터 중 하나로 데이터 영역을 의미한다. 크기는 2바이트이다. dword ptr은 메모리에 있는 데이터를 나타낸다. 이때 나타내는 데이터의 크기는 4바이트이다.*) 해당 명령어에 브레이크 포인트를 걸어두고 다시 실행하여 시리얼 값으로 16을 입력한다. 그러면 `eax`에 10이 저장됨을 확인할 수 있다.



<시리얼 값으로 16을 입력 후, 변환된 eax 값>

내가 입력한 값을 16진수로 변환시키고 `ds:[45B844]` 데이터와 비교한 뒤 같으면 성공 메시지를 출력하는 것으로 추측된다. `ds:[45B844]` 데이터를 확인하기 위해 15.0045B844의 덤프를 따라간다. 그러면 해당 데이터가 0x6160임을 알 수 있다.

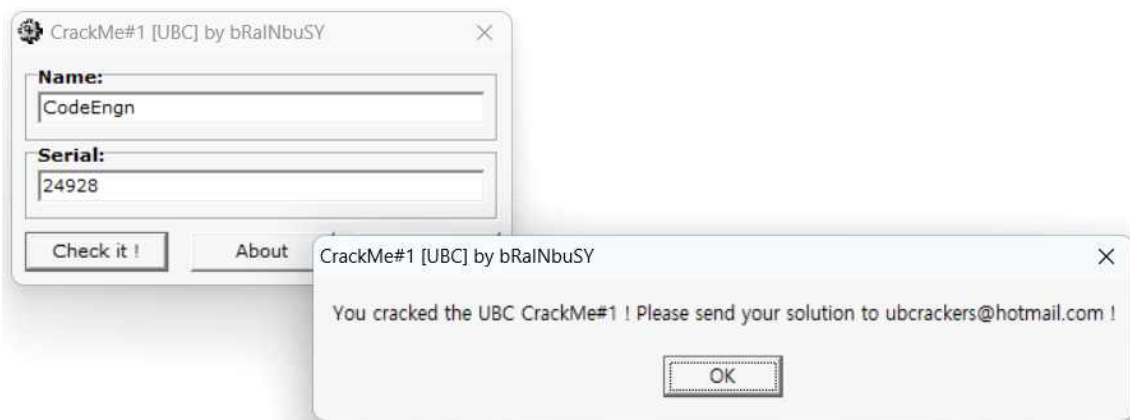


<ds:[45B844] 덤프 따라가기>

Address	Hex
0045B844	60 61 00 00 09 00 00 00 FF FF FF FF 00 00 00 00
0045B854	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0045B864	0C [0045B844] = 00006160 (User Data) 00 00 00 00
0045B874	0C 00 00 00 00 00 00 00 00 00 00 00 00 00
0045B884	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0045B894	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0045B8A4	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0045B8B4	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0045B8C4	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0045B8D4	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0045B8E4	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0045B8F4	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0045B904	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0045B914	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0045B924	00 00 00 00 00 00 00 00 00 00 00 00 00 00
0045B934	00 00 00 00 00 00 00 00 00 00 00 00 00 00

<ds:[45B844] 데이터 확인(리틀엔디안)>

0x6160을 10진수로 변환하여 시리얼 값으로 입력해보면 성공 메시지를 확인할 수 있다.



<성공 메시지 출력>

답 : 24928