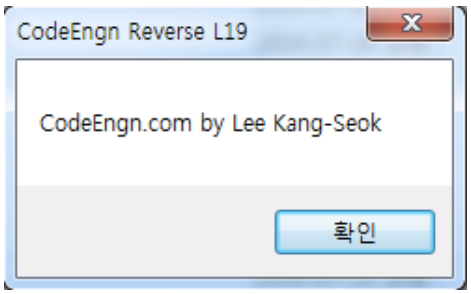
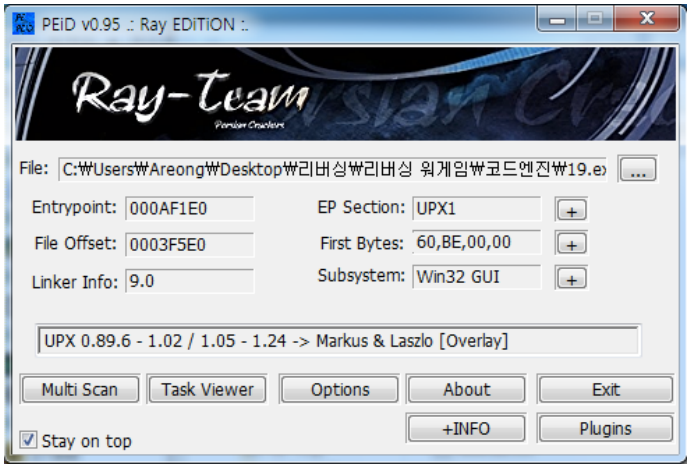
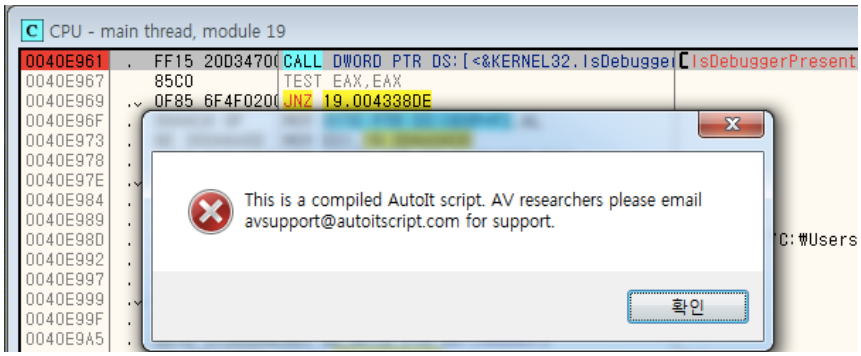


프로그램을 실행하면 메세지 박스가 출력되고 10초 정도 지나면 프로그램이 종료됩니다.



패킹이 되어있기 때문에 언패킹 후 진행하겠습니다.



분석을 진행하다 보면 IsDebuggerPresent를 통해 디버깅을 감지하고 위와 같은 에러메세지를 출력합니다.

디버깅 우회는 이전 문제에서 다뤘으니 생략하도록 하겠습니다.

```
0040E45F  CALL DWORD PTR DS:[<&USER32.SetTimer>]  USER32.SetTimer
0040E523  CALL DWORD PTR DS:[<&USER32.SetTimer>]  USER32.SetTimer
0044646E  CALL DWORD PTR DS:[<&USER32.SetTimer>]  USER32.SetTimer
0046D8B7  CALL DWORD PTR DS:[<&USER32.SetTimer>]  USER32.SetTimer
```

Found intermodular calls

Address	Disassembly	Destination
0040B350	CALL DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime
0040E6CA	CALL DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime
004301F3	CALL DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime
004305BC	CALL DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime
0043197F	CALL DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime
00431D70	CALL DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime
00431EED	CALL DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime
00444C44	CALL EDI	WINMM.timeGetTime
00451B82	CALL DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime
00451B9E	CALL DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime
00456F68	CALL DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime
0046FBC1	CALL DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime
0046FBDC	CALL DWORD PTR DS:[<&WINMM.timeGetTime>]	WINMM.timeGetTime

프로그램이 특정 시간이 지난 뒤에 종료되기 때문에 시간 관련 함수를 찾아보겠습니다.
SetTimer와 timeGetTime 두 함수에 전부 BreakPoint를 설정했습니다.

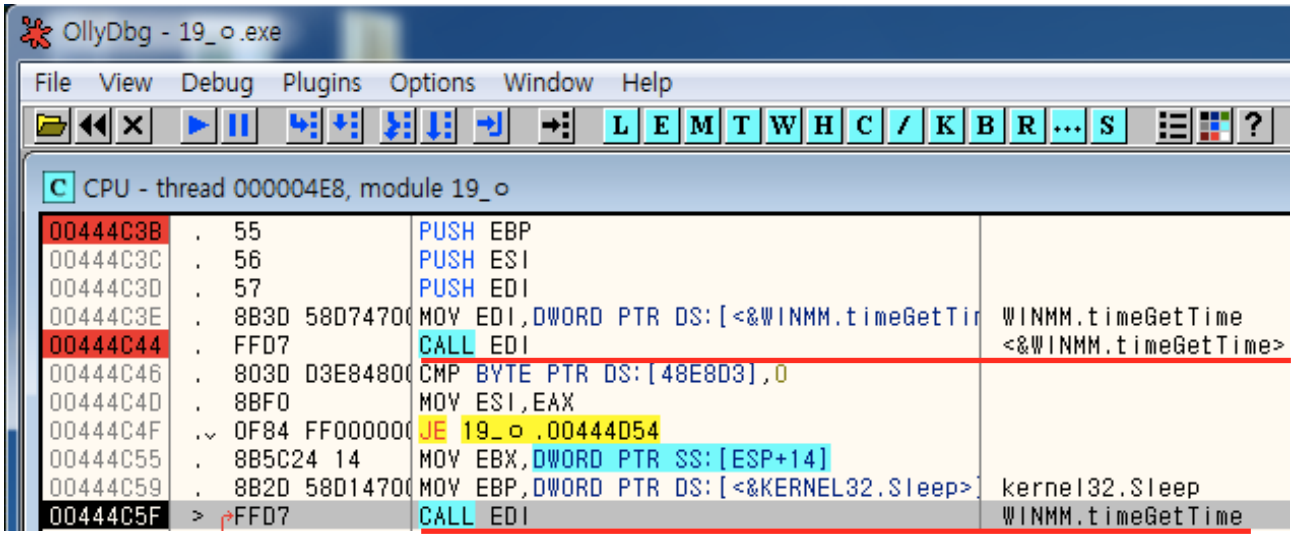
SetTimer는 타이머를 설정해서 일정 시간이 지나면 원하는 동작을 처리할 수 있는 함수입니다.

```
0040E519  >  6A 00        PUSH 0                  Timerproc = NULL; Case 1 (WM_CREATE) of switch 0040E4DB
0040E51B  .  68 EE020000  PUSH 2EE                Timeout = 750. ms
0040E520  .  6A 01        PUSH 1                  TimerID = 1
0040E522  .  57           PUSH EDI                hWnd
0040E523  .  FF15 BCD6470( CALL DWORD PTR DS:[<&USER32.SetTimer>]  SetTimer
```

timeGetTime은 프로그램이 실행된 시간이 얼마나 지났는지 알려주는 함수입니다.

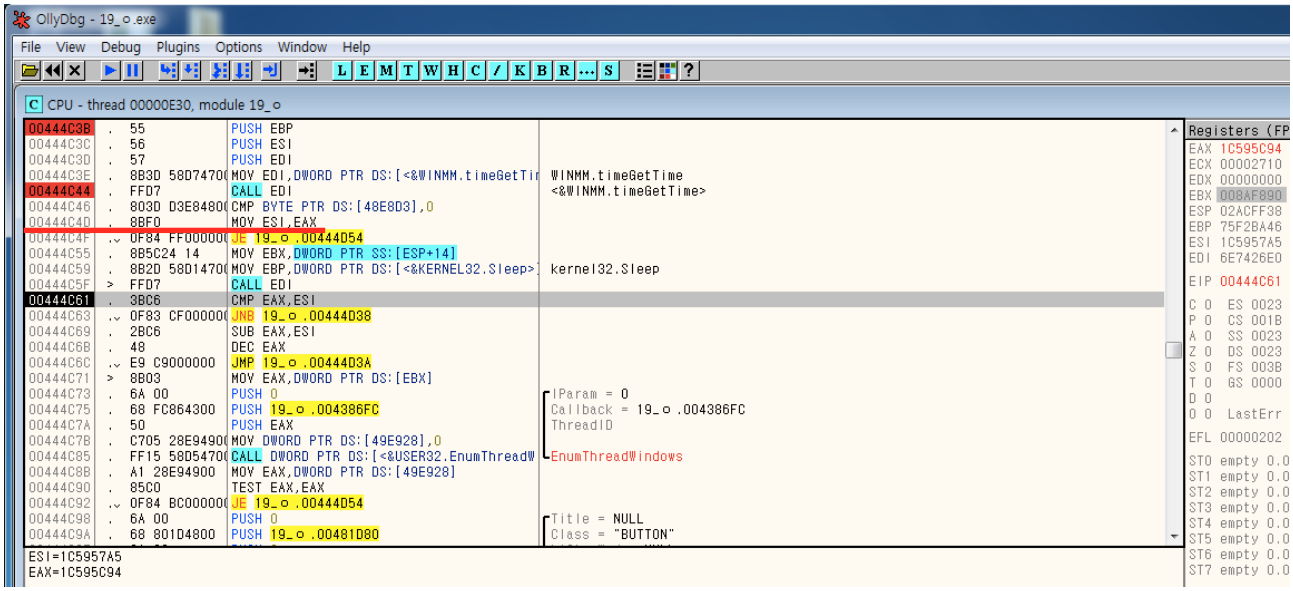
실행해 보면 SetTimer에 걸리게 되는데 설정된 시간이 750ms 인것을 볼 수 있습니다.

하지만 프로그램이 10초 정도 시간이 지난 뒤 종료되는 것으로 보면 SetTimer를 이용한 것이 아닌것 같습니다.



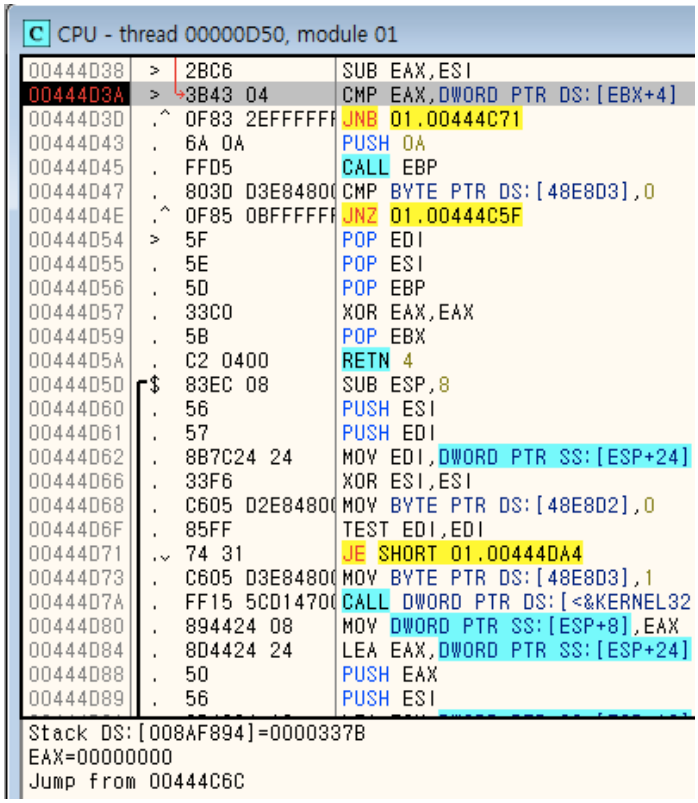
timeGetTime 함수 부분인데 EDI에 해당 함수를 넣고 두 번 호출하는 것을 알 수 있습니다.

각각의 함수가 호출된 뒤에는 EAX에 리턴값이 저장됩니다.



첫번째 호출의 결과는 ESI에 저장되게 되고 두번째 호출은 EAX에 저장되어 서로 비교하게 됩니다.

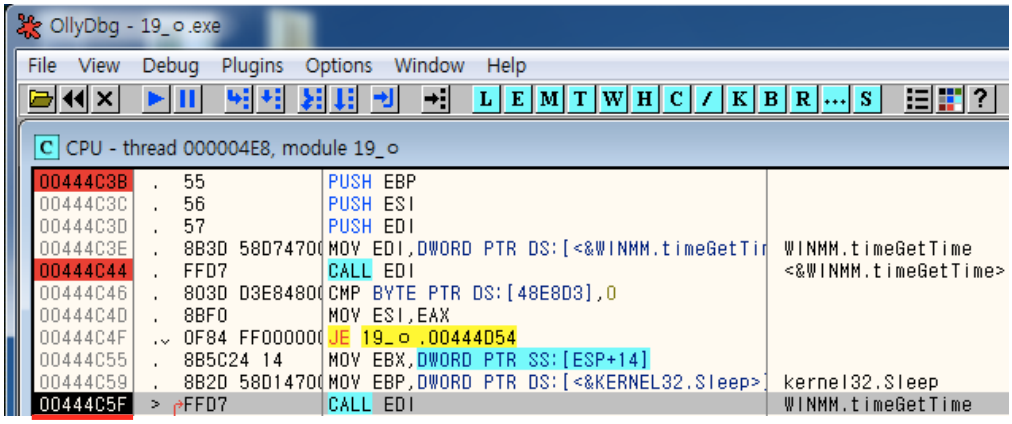
비교한 뒤 JNB(Jump Not Below)를 통해 분기하는데 EAX가 나중에 호출된 결과이므로 점프를 진행합니다.



점프 뒤에 EAX에서 ESI를 빼게 됩니다. 그리고 그 결과를 EBX+4와 비교하게 됩니다.

지금은 스샷을 찍느라 EAX의 값이 더 크지만 프로그램 흐름대로라면 EAX의 값은 EBX+4의 값보다 작습니다.

그 후 00444D4E주소의 점프를 통해 두번째 호출로 점프하게 됩니다.



점프문의 분기 장소와 두 번째 호출의 주소가 같음을 알 수 있습니다.

첫 번째 호출은 프로그램이 시작된 시각이 되고 두 번째 호출은 시작된 뒤의 시간을 지속적으로 받아옵니다.

이 두 호출의 결과를 빼게 되면 프로그램이 실행된 시각을 알 수 있고 그 값과 EBX+4의 값과 비교해서

크거나 같으면 프로그램이 종료됩니다.