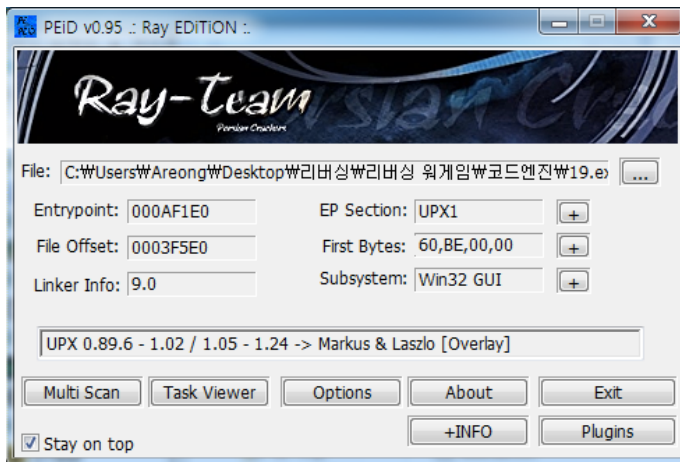
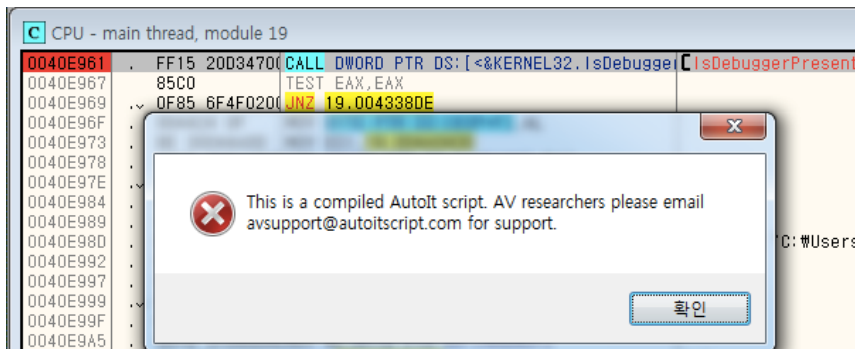




프로그램을 실행하면 메세지 박스가 출력되고 10초 정도 지나면 프로그램이 종료됩니다.



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

R 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 Timeout = 750. ms TimerID = 1 hWnd SetTimer
0040E51B	. 68 EE020000	PUSH 2EE	
0040E520	. 6A 01	PUSH 1	
0040E522	. 57	PUSH EDI	
0040E523	. FF15 8CD64700	CALL DWORD PTR DS: [<&USER32.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.

```

CPU - thread 000004E8, module 19_o
00444C3B . 55      PUSH EBP
00444C3C . 56      PUSH ESI
00444C3D . 57      PUSH EDI
00444C3E . 8B3D 58D74700 MOV EDI,DWORD PTR DS:[<&WINMM.timeGetTime]
00444C44 . FF07     CALL EDI
00444C46 . 803D D3E84800 CMP BYTE PTR DS:[48E8D3],0
00444C4D . 8BF0     MOV ESI,EAX
00444C4F . 0F84 FF000000 JE 19_o.00444D54
00444C55 . 8B5C24 14 MOV EBX,DWORD PTR SS:[ESP+14]
00444C59 . 8B2D 58D14700 MOV EBP,DWORD PTR DS:[<&KERNEL32.Sleep>]
00444C5F . FF07     CALL EDI

```

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

각각의 함수가 호출된 뒤에는 EAX에 리턴값이 저장됩니다.

```

CPU - thread 00000E30, module 19_o
00444C3B . 55      PUSH EBP
00444C3C . 56      PUSH ESI
00444C3D . 57      PUSH EDI
00444C3E . 8B3D 58D74700 MOV EDI,DWORD PTR DS:[<&WINMM.timeGetTime]
00444C44 . FF07     CALL EDI
00444C46 . 803D D3E84800 CMP BYTE PTR DS:[48E8D3],0
00444C4D . 8BF0     MOV ESI,EAX
00444C4F . 0F84 FF000000 JE 19_o.00444D54
00444C55 . 8B5C24 14 MOV EBX,DWORD PTR SS:[ESP+14]
00444C59 . 8B2D 58D14700 MOV EBP,DWORD PTR DS:[<&KERNEL32.Sleep>]
00444C5F . FF07     CALL EDI
00444C61 . 3BC6     CMP EAX,ESI
00444C63 . 0F83 CF000000 JNB 19_o.00444D38
00444C69 . 2BC6     SUB EAX,ESI
00444C6B . 48       DEC EAX
00444C6C . E9 C9000000 JMP 19_o.00444D3A
00444C71 . 8B03     MOV EAX,DWORD PTR DS:[EBX]
00444C73 . 6A 00    PUSH 0
00444C75 . 68 FC864300 PUSH 19_o.004386FC
00444C7A . 50       PUSH EAX
00444C7B . C705 28E94900 MOV DWORD PTR DS:[49E928],0
00444C85 . FF15 58D54700 CALL DWORD PTR DS:[<&USER32.EnumThreadWindows>]
00444C8B . A1 28E94900 MOV EAX,DWORD PTR DS:[49E928]
00444C90 . 85C0     TEST EAX,EAX
00444C92 . 0F84 BC000000 JE 19_o.00444D54
00444C98 . 6A 00    PUSH 0
00444C9A . 68 801D4800 PUSH 19_o.00481D80

```

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.

```

CPU - thread 00000D50, module 01
00444D38 > 2BC6 SUB EAX,ESI
00444D3A > 3B43 04 CMP EAX,DWORD PTR DS:[EBX+4]
00444D3D ^ 0F83 2EFFFFFF JNB 01.00444C71
00444D43 . 6A 0A PUSH 0A
00444D45 . FFD5 CALL EBP
00444D47 . 803D D3E84800 CMP BYTE PTR DS:[48E8D3],0
00444D4E ^ 0F85 0BFFFFFF JNZ 01.00444C5F
00444D54 > 5F POP EDI
00444D55 . 5E POP ESI
00444D56 . 5D POP EBP
00444D57 . 33C0 XOR EAX,EAX
00444D59 . 5B POP EBX
00444D5A . C2 0400 RETN 4
00444D5D $ 83EC 08 SUB ESP,8
00444D60 . 56 PUSH ESI
00444D61 . 57 PUSH EDI
00444D62 . 8B7C24 24 MOV EDI,DWORD PTR SS:[ESP+24]
00444D66 . 33F6 XOR ESI,ESI
00444D68 . C605 D2E84800 MOV BYTE PTR DS:[48E8D2],0
00444D6F . 85FF TEST EDI,EDI
00444D71 ^ 74 31 JE SHORT 01.00444DA4
00444D73 ^ 0F85 D3E84800 MOV BYTE PTR DS:[48E8D3],1
00444D7A . FF15 5CD14700 CALL DWORD PTR DS:[<&KERNEL32
00444D80 . 894424 08 MOV DWORD PTR SS:[ESP+8],EAX
00444D84 . 8D4424 24 LEA EAX,DWORD PTR SS:[ESP+24]
00444D88 . 50 PUSH EAX
00444D89 . 56 PUSH ESI

Stack DS:[008AF894]=0000337B
EAX=00000000
Jump from 00444C6C

```

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.

```

OllyDbg - 19_o.exe
File View Debug Plugins Options Window Help
[Icons] [L] [E] [M] [T] [W] [H] [C] / [K] [B] [R] ... [S] [Icons] [?]

CPU - thread 000004E8, module 19_o
00444C3B . 55 PUSH EBP
00444C3C . 56 PUSH ESI
00444C3D . 57 PUSH EDI
00444C3E . 8B3D 58D74700 MOV EDI,DWORD PTR DS:[<&WINMM.timeGetTime] WINMM.timeGetTime
00444C44 . FFD7 CALL EDI <&WINMM.timeGetTime>
00444C46 . 803D D3E84800 CMP BYTE PTR DS:[48E8D3],0
00444C4D . 8BF0 MOV ESI,EAX
00444C4F ^ 0F84 FF000000 JE 19_o.00444D54
00444C55 . 8B5C24 14 MOV EBX,DWORD PTR SS:[ESP+14]
00444C59 . 8B2D 58D14700 MOV EBP,DWORD PTR DS:[<&KERNEL32.Sleep>] kernel32.Sleep
00444C5F > FFD7 CALL EDI WINMM.timeGetTime

```

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.