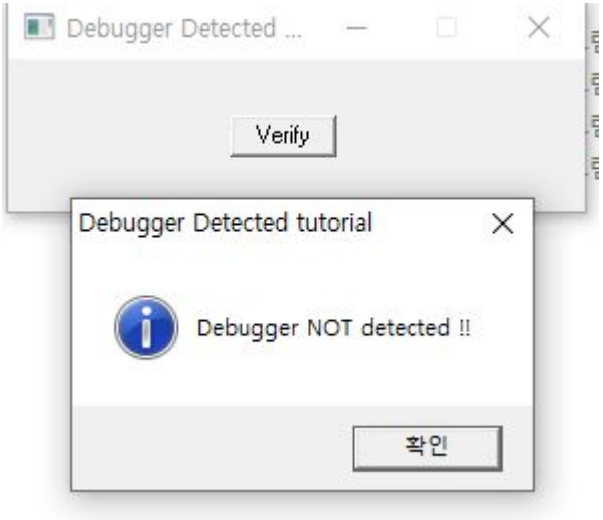
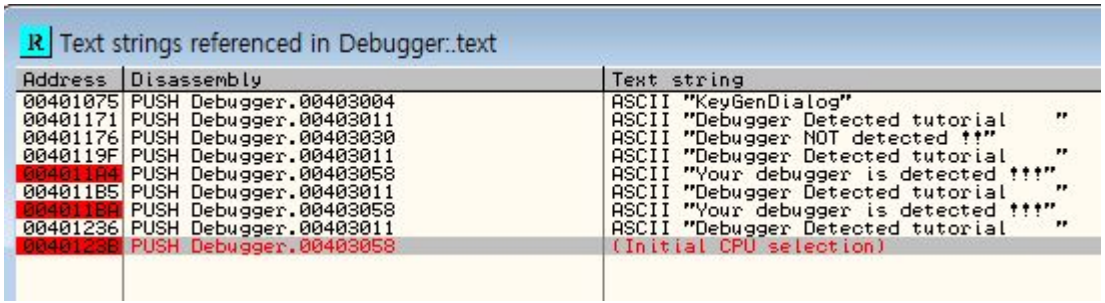
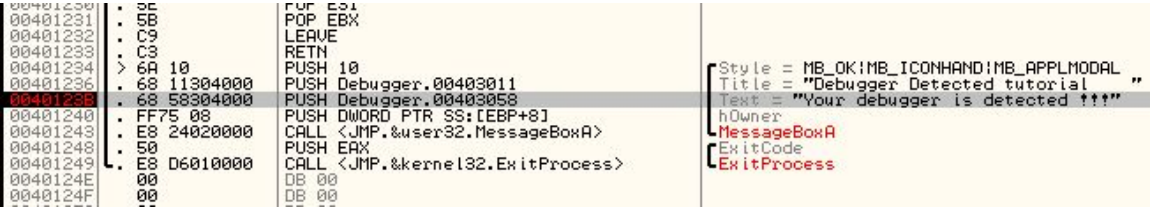



```
00401231 | . 5B            | POP EBX
00401232 | . C9            | LEAVE
00401233 | . C3            | RETN
00401234 | > 6A 10         | PUSH 10                                  | Style = MB_OK|MB_ICONHAND|MB_APPLMODAL
00401236 | . 68 11304000   | PUSH Debugger.00403011                   | Title = "Debugger Detected tutorial      "
0040123B | . 68 58304000   | PUSH Debugger.00403058                   | Text = "Your debugger is detected !!!"
00401240 | . FF75 08       | PUSH DWORD PTR SS:[EBP+8]                | hOwner
00401243 | . E8 24020000   | CALL <JMP.&user32.MessageBoxA>           | MessageBoxA
00401248 | . 50            | PUSH EAX                                 | ExitCode
00401249 | . E8 D6010000   | CALL <JMP.&kernel32.ExitProcess>         | ExitProcess
0040124E |   00            | DB 00
0040124F |   00            | DB 00
00401250 |   00            | DB 00
00401251 |   00            | DB 00
```

튜토리얼 19의 첫번째 분석 프로그램이다. 프로그램을 실행하면 디버거가 탐지 되지 않았다고 한다. ollydbg로 실행시켰을 때는 디버거가 탐지됐다고 나온다.

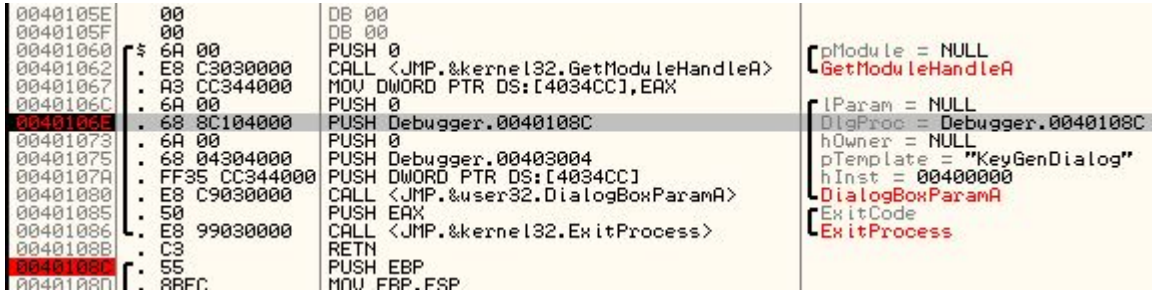
이 부분을 우회해 디버거로 분석해도 해당 오류가 뜨지 않도록 한다.

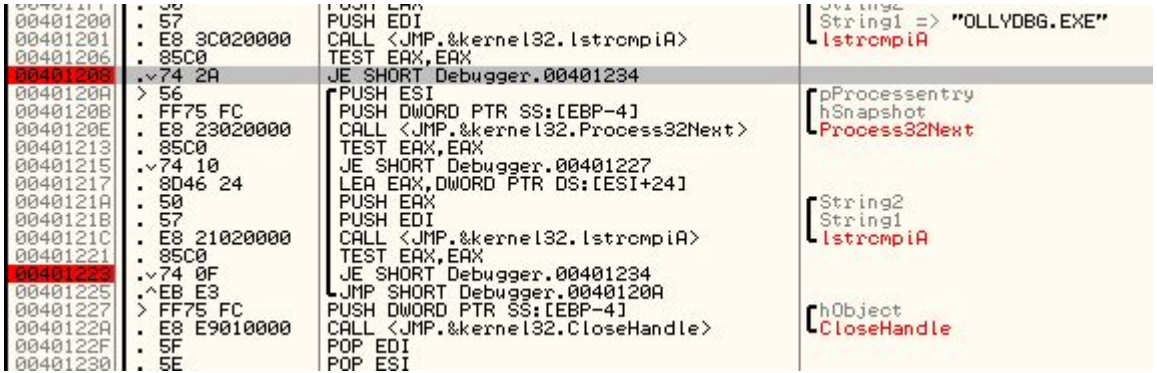




텍스트를 보면 디버거가 탐지됐다고 나온 부분이 세개 보인는데, 이 세 부분에 모두 bp를 걸고 실행한다.

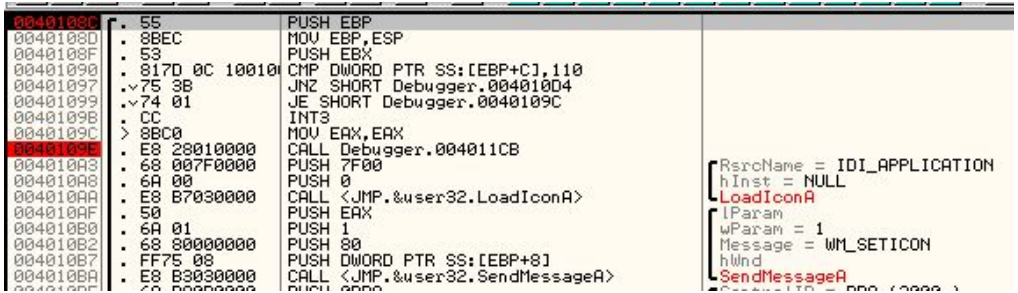
그럼 제일 마지막 호출에서 멈춘다. 그리고 바로 밑에서 프로그램을 종료한다.

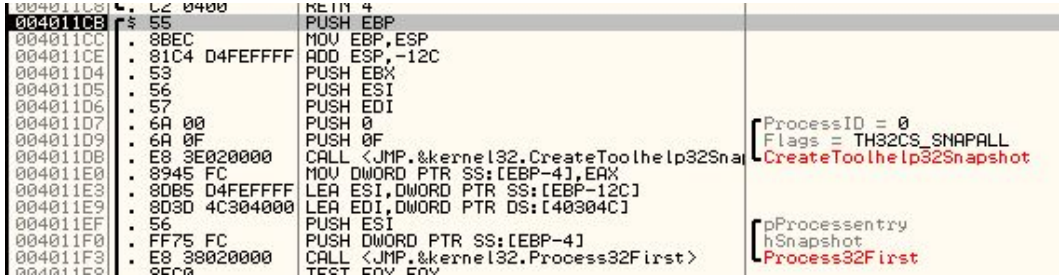




해당 부분을 이 두 부분에서 호출하고 있다.

프로그램을 시작하면 DialogBoxParamA에서 해당 부분을 호출한다. DlgProc에 메세지를 띄울 함수 주소를 넣고, 해당 함수에서 디버거 탐지 여부에 따라 메시지를 띄운다.

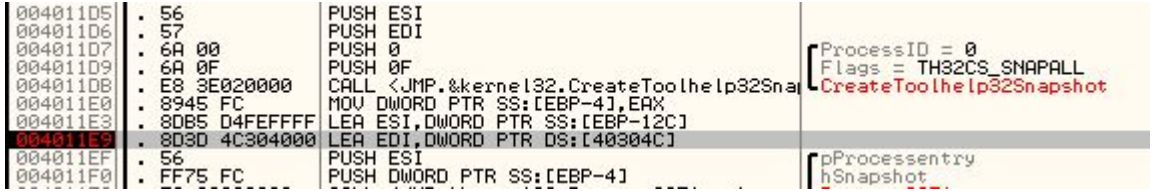




프로그램을 실행하다보면 40109E 까지 실행되고, SendMessageA를 호출하면 다시 40108C부분으로 가게 된다. 4011CB 함수를 분석해 봐야할 것 같다.

함수 내부에서 첫 함수로 CreateToolhelp32SnapShot함수를 호출하고 있다.

이 함수는 특정 프로세스의 힙, 쓰레드, 모듈 등의 상태를 리턴한다. 즉, 해당 프로세스의 핸들값을 리턴한다.



함수를 호출하고, EDI에 어떤값을 옮기는지 보니 OOOYDBG.EXE 문자열을 옮기고 있다.

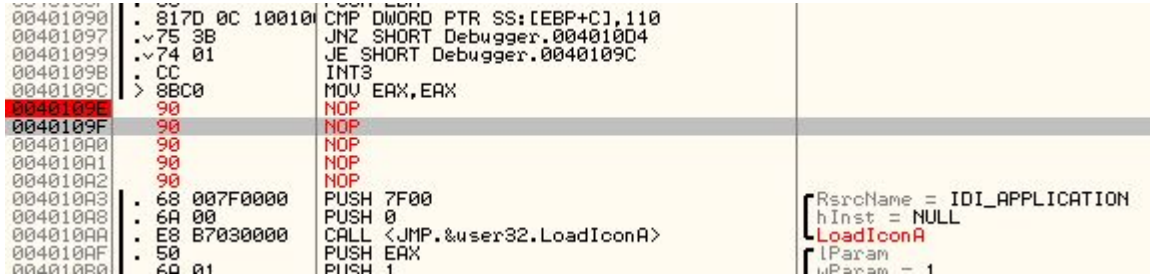
그런다음 Process32First 함수를 호출하는데, 현재 실행되고 있는 프로세스 목록중 첫번째 프로세스를 반환한다.

```
004011F3  . E8 38020000   CALL <JMP.&kernel32.Process32First>     Process32First
004011F8  . 85C0          TEST EAX,EAX
004011FA  .v74 2B         JE SHORT Debugger.00401227
004011FC  . 8D46 24       LEA EAX,DWORD PTR DS:[ESI+24]
004011FF  . 50            PUSH EAX                                String2
00401200  . 57            PUSH EDI                                String1 => "OLLYDBG.EXE"
00401201  . E8 3C020000   CALL <JMP.&kernel32.lstrcmpiA>          lstrcmpiA
00401206  . 85C0          TEST EAX,EAX
00401208  .v74 2A         JE SHORT Debugger.00401234
0040120A  > 56            PUSH ESI                                pProcessentry
0040120B  . FF75 FC       PUSH DWORD PTR SS:[EBP-4]               hSnapshot
0040120E  . E8 23020000   CALL <JMP.&kernel32.Process32Next>      Process32Next
00401213  . 85C0          TEST EAX,EAX
00401215  .v74 10         JE SHORT Debugger.00401227
```

```
004011FC  . 8D46 24       LEA EAX,DWORD PTR DS:[ESI+24]
004011FF  . 50            PUSH EAX                                String2
00401200  . 57            PUSH EDI                                String1 => "OLLYDBG.EXE"
00401201  . E8 3C020000   CALL <JMP.&kernel32.lstrcmpiA>          lstrcmpiA
00401206  . 85C0          TEST EAX,EAX
00401208  .v74 2A         JE SHORT Debugger.00401234
0040120A  > 56            PUSH ESI                                pProcessentry
0040120B  . FF75 FC       PUSH DWORD PTR SS:[EBP-4]               hSnapshot
0040120E  . E8 23020000   CALL <JMP.&kernel32.Process32Next>      Process32Next
00401213  . 85C0          TEST EAX,EAX
00401215  .v74 10         JE SHORT Debugger.00401227
00401217  . 8D46 24       LEA EAX,DWORD PTR DS:[ESI+24]
0040121A  . 50            PUSH EAX                                String2
0040121B  . 57            PUSH EDI                                String1
0040121C  . E8 21020000   CALL <JMP.&kernel32.lstrcmpiA>          lstrcmpiA
00401221  . 85C0          TEST EAX,EAX
00401223  .v74 0F         JE SHORT Debugger.00401234
00401225  .^EB E3         JMP SHORT Debugger.0040120A
00401227  > FF75 FC       PUSH DWORD PTR SS:[EBP-4]               hObject = 00000194 (window)
0040122A  . E8 E9010000   CALL <JMP.&kernel32.CloseHandle>        CloseHandle
0040122F  . 5F            POP EDI
00401230  . 5E            POP ESI
```

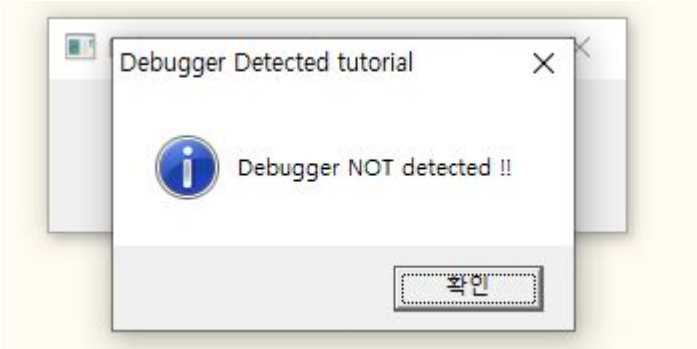
반환된 프로세스명과
OLLYDBG.EXE을 비교한다.

그리고 다음에 어떤 루프가
나타나는데, Process32Next를
통해서 다음 프로세스를 호출한다.
즉, 프로세스마다 이름을 비교해
OLLYDBG.EXE 함수가 있는지
확이하는 것이다. 조건이 충족되면
아까 디버거가 탐지됐다는 문자열이
있는곳으로 분기된다.



방금 분석한 함수는 오로지 오류 메시지를 호출하는 루틴이다. 그래서 아까 해당 부분으로 호출하는 함수 부분을 호출하지 못하도록 패치하면 된다.

```
00401151  $ 55              PUSH EBP
00401152  . 8BEC            MOV EBP,ESP
00401154  . 90              NOP
00401155  . 90              NOP
00401156  . 90              NOP
00401157  . 90              NOP
00401158  . 90              NOP
00401159  . BF D0344000     MOV EDI,Debugger.004034D0
0040115E  . BE F4344000     MOV ESI,Debugger.004034F4
00401163  . 8A07            MOV AL,BYTE PTR DS:[EDI]
00401165  . 8A0E            MOV CL,BYTE PTR DS:[ESI]
00401167  . 0AC0            OR AL,AL
00401169  .v74 04           JE SHORT Debugger.0040116F
0040116B  . 0AC9            OR CL,CL
0040116D  .v75 16           JNZ SHORT Debugger.00401185
0040116F  > 6A 40           PUSH 40                          Style = MB_OK|MB_ICONASTERISK|MB_APPLMODAL
00401171  . 68 11304000     PUSH Debugger.00403011           Title = "Debugger Detected tutorial     "
00401176  . 68 30304000     PUSH Debugger.00403030           Text = "Debugger NOT detected !!"
0040117B  . FF75 08         PUSH DWORD PTR SS:[EBP+8]        hOwner
0040117E  . E8 E9020000     CALL <JMP.&user32.MessageBoxA>   MessageBoxA
00401183  .vEB 42           JMP SHORT Debugger.004011C7
00401185  > E8 0A010000     CALL Debugger.00401294
0040118A  . 68 F4344000     PUSH Debugger.004034F4           String2 = ""
```



그리고 디버거가 탐지 되지 않았다는 메시지를 호출하는 함수 부분에서도 오류메시지로 호출하는 코드가 있어 패치해 주었고, ollydbg로 실행해도 디버거가 탐지 되지 않았다고 나온다.