

# 악성코드 분석 보고서

(sand-reversingwithlana-tutorials)

2025.08.06

## 1. 문제

### 1. ReverseMe.A

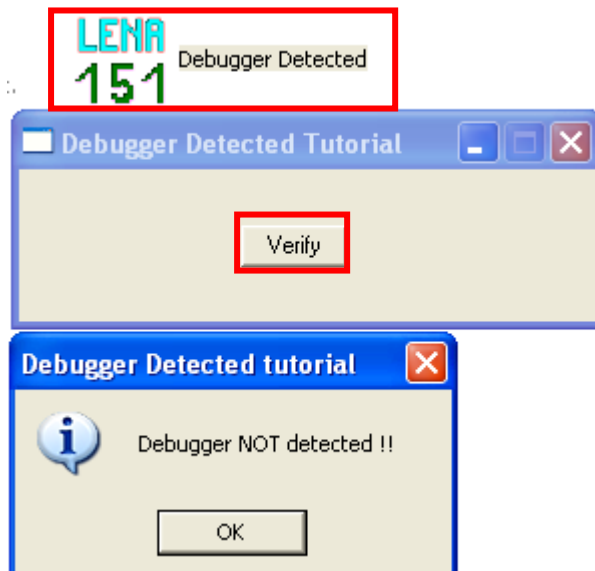


클릭 시 성공했다는 창이 나온다.

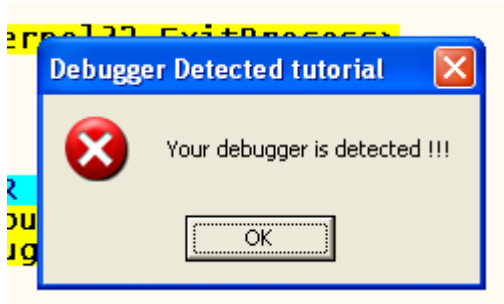


올리디버그 실행 시 키 파일이 유효하지 않다고 나온다.

### 2. Debugger Detected



'Debugger Detected'를 누르면 밑에 창이 나오고 'Verify'를 누르면 'Debuuger NOT detected' 창이 나온다.



하지만 올리디버그로 실행 시 구문이 다른 창이 나오는 걸 알 수 있다.

왜 서로 창이 다르게 나오는 지 알아봐야한다.

## 2. 해결 방법

### 1. ReverseMe.A

```

00401057 . A3 AF214000 MOV DWORD PTR DS:[4021AF],EAX
0040105C . 6A 00      PUSH 0
0040105E . 68 6F214000 PUSH ReverseM.0040216F
00401063 . 6A 03      PUSH 3
00401065 . 6A 00      PUSH 0
00401067 . 6A 03      PUSH 3
00401069 . 68 000000C0 PUSH C0000000
0040106E . 68 79204000 PUSH ReverseM.00402079
00401073 . E8 0B020000 CALL <JMP.&kernel32.CreateFileA>
00401078 . 83FB FF    CMP EAX,-1
0040107B . 75 1D      JNZ SHORT ReverseM.0040109A
0040107D . 6A 00      PUSH 0
0040107F . 68 00204000 PUSH ReverseM.00402000
00401084 . 68 17204000 PUSH ReverseM.00402017
00401089 . 6A 00      PUSH 0
0040108B . E8 D7020000 CALL <JMP.&user32.MessageBoxA>
00401090 . FR 74020000 CALL <JMP.&kernel32.ExitProcess>

```

hTemplateFile = NULL  
Attributes = READONLY|HIDDEN|SYSTEM|ARCHIVE|TE  
Mode = OPEN\_EXISTING  
pSecurity = NULL  
ShareMode = FILE\_SHARE\_READ|FILE\_SHARE\_WRITE  
Access = GENERIC\_READ|GENERIC\_WRITE  
FileName = "keyfile.dat"  
CreateFileA  
Style = MB\_OK|MB\_APPLMODAL  
Title = "Key File ReverseMe"  
Text = "Evaluation period out of date. Purchas  
MessageBoxA

Keyfile.dat라는 파일을 여는 함수를 볼 수 있는데 지금 Keyfile.dat이 있기 때문에 레지스터 키 값을 구매해달라는 메시지 박스는 넘어가는 걸 알 수 있다.

```

0040109A > 6A 00      PUSH 0
0040109C . 68 73214000 PUSH ReverseM.00402173
004010A1 . 6A 46      PUSH 46
004010A3 . 68 1A214000 PUSH ReverseM.0040211A
004010A8 . 50        PUSH EAX
004010A9 . E8 2F020000 CALL <JMP.&kernel32.ReadFile>
004010AE . 85C0      TEST EAX,EAX

```

pOverlapped = NULL  
pBytesRead = ReverseM.00402173  
BytesToRead = 46 (70.)  
Buffer = ReverseM.0040211A  
hFile  
ReadFile

ReadFile함수가 실행되는데 keyfile.dat의 값들이 0x0040211A에 저장되는 걸 볼 수 있다.

\* pBytesRead와 Buffer의 차이

pBytesRead : 함수 호출 후 실제 읽혀진 데이터의 크기를 저장할 메모리의 주소

Buffer : 읽어온 내용을 저장할 메모리의 시작 주소를 설정한다.

Address	Hex dump	ASCII
0040211A	47 47 47 47 47 47 47 47	GGGGGGGG
00402122	47 47 30 30 30 30 30 30	GG000000
0040212A	30 30 30 30 30 30 30 00	0000000.
00402132	00 00 00 00 00 00 00 00	.....
0040213A	00 00 00 00 00 00 00 00	.....

해당 덤프 창에 들어가서 보면 값이 저장된 걸 볼 수 있다.

```

004010C1 > 8A83 1A214000 MOV AL,BYTE PTR DS:[EBX+40211A]
004010C7 . 3C 00      CMP AL,0
004010C9 . 74 08      JE SHORT ReverseM.004010D3
004010CB . 3C 47      CMP AL,47
004010CD . 75 01      JNZ SHORT ReverseM.004010D0
004010CF . 46        INC ESI
004010D0 . 43        INC EBX
004010D1 . EB EE      JMP SHORT ReverseM.004010C1
004010D3 > EB 23000000 CALL ReverseM.004010FB

```

이 부분은 keyfile.dat에서 'G'문자가 몇 개인지 세는 과정이다.

```

004010D3 > EB 23000000 CALL ReverseM.004010FB
004010D8 . 83FE 08    CMP ESI,8
004010DB . 7C 05      JL SHORT ReverseM.004010E1
004010DD . E8 2C000000 CALL ReverseM.0040110E
004010E2 > 6A 00      PUSH 0
004010E4 . 68 00204000 PUSH ReverseM.00402000
004010E9 . 68 86204000 PUSH ReverseM.00402086

```

Key File ReverseMe  
Keyfile is not valid. Sorry.  
OK

이 부분에서 창이 나오는 걸 볼 수 있고 'F7'을 눌러 함수에 들어가본다.

004010DD	. E8 2C000000	CALL ReverseM.0040110E	
004010E2	> 6A 00	PUSH 0	
004010E4	. 68 00204000	PUSH ReverseM.00402000	
004010E9	. 68 86204000	PUSH ReverseM.00402086	
004010EE	. 6A 00	PUSH 0	
004010F0	. E8 72020000	CALL <JMP.&user32.MessageBoxA>	[Style = MB_OK MB_APPLMODAL
004010F5	. E8 BF010000	CALL <JMP.&kernel32.ExitProcess>	Title = " Key File ReverseMe"
004010FA	. C3	RETN	Text = "Keyfile is not valid. Sorry"
004010FB	\$ E8 D7010000	CALL <JMP.&kernel32.IsDebuggerPresent>	hOwner = NULL
00401100	. 83F8 01	CMP EAX,1	MessageBoxA
00401103	. ^ 74 DD	JE SHORT ReverseM.004010E2	ExitProcess
00401105	. C3	RETN	

IsDebuggerPresent함수로 넘어간 후 MessageBoxA로 넘어가는 걸 알 수 있다. 메시지 박스로 넘어가기 전에 EAX를 1과 비교하는 걸 볼 수 있는데 IsDebuggerPresent함수가 호출 프로세스가 사용자 모드 디버거에 의해 디버그되고 있는지 여부를 확인하는 함수라고 나와있고

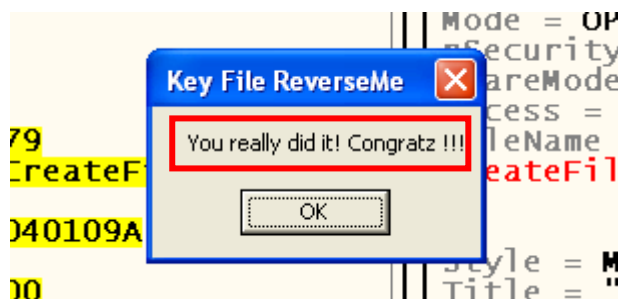
반환 값으로는 현재 프로세스가 디버거의 컨텍스트에서 실행 중인 경우 반환 값은 0이 아닌 값, 실행되고 있지 않으면 반환 값은 0이다.

하지만 올리디버거에서 실행되고 있으므로 반환 값은 0이 아닌 값이고 레지스터를 보면 EAX = 1로 나오는 걸 볼 수 있다.

그럼 성공했다는 창이 나오기 위해서는 EAX = 0으로 되거나 JE -> JNE로 바꿔야 성공 창이 나오는 걸 알 수 있다.

004010FB	\$ E8 D7010000	CALL <JMP.&kernel32.IsDebuggerPresent>	[IsDebuggerPresent
00401100	. 83F8 01	CMP EAX,1	
00401103	. ^ 75 DD	JNZ SHORT ReverseM.004010E2	
00401105	. C3	RETN	
00401106	. 6A 00	PUSH 0	[ExitCode = 0
00401108	. E8 AC010000	CALL <JMP.&kernel32.ExitProcess>	ExitProcess

JE 004010E2 -> JNE 004010E2로 바꾸고 저장하고 올리디버거로 실행해보면



해당 창이 나오는 걸 볼 수 있다.

문제 해결!

## 2. Debugger Detected

00401060	6A 00	PUSH 0	pModule = NULL
00401062	E8 C3030000	CALL <JMP.&kernel32.GetModuleHandleA>	GetModuleHandleA
00401067	A3 CC344000	MOV DWORD PTR DS:[4034CC],EAX	
0040106C	6A 00	PUSH 0	lParam = NULL
0040106E	68 8C104000	PUSH Debugger.0040108C	DlgProc = Debugger.0040108C
00401073	6A 00	PUSH 0	hOwner = NULL
00401075	68 04304000	PUSH Debugger.00403004	pTemplate = "KeyGenDia"
0040107A	FF35 CC344000	PUSH DWORD PTR DS:[4034CC]	hInst = 00400000
00401080	E8 C9030000	CALL <JMP.&user32.ShowDialogParamA>	DialogBoxParamA
00401085	50	PUSH EAX	ExitCode
00401086	E8 99030000	CALL <JMP.&kernel32.ExitProcess>	ExitProcess

0x00401080에서 실행 시 창이 나오는 걸 알 수 있다.

DialogBoxParamA 를 msdn 에서 찾아보면 DLGPROC 인자가 대화 상자 프로시저에 대한 포인터라고 나온다. 즉, 0x0040108C 에 있는 함수가 실행된다는 뜻이다.

00401085	50	PUSH EAX	
00401086	E8 99030000	CALL <JMP.&kernel32.ExitProcess>	
0040108B	C3	RETN	
0040108C	55	PUSH EBP	
0040108D	8BEC	MOV EBP,ESP	
0040108F	53	PUSH EBX	
00401090	817D 0C 10010000	CMP DWORD PTR SS:[EBP+C],110	
00401097	75 3B	JNZ SHORT Debugger.004010D4	
00401099	74 01	JE SHORT Debugger.0040109C	
0040109B	CC	INT3	

0x0040108C 에 BP 를 걸고 'ctrl+F2'로 재실행 후 해당 주소로 넘어오고 'F8'로 실행해보면 0x00401097 에서 점프하는 걸 알 수 있다.

0040108C	55	PUSH EBP	
0040108D	8BEC	MOV EBP,ESP	
0040108F	53	PUSH EBX	
00401090	817D 0C 10010000	CMP DWORD PTR SS:[EBP+C],110	
00401097	75 3B	JNZ SHORT Debugger.004010D4	
00401099	74 01	JE SHORT Debugger.0040109C	
0040109B	CC	INT3	
0040109C	8BC0	MOV EAX,EAX	
0040109E	E8 28010000	CALL Debugger.004011CB	
004010A3	68 007F0000	PUSH 7F00	
004010A8	6A 00	PUSH 0	

계속 실행 해보면 다시 0x0040108C 여기에 도달하는 걸 알 수 있고 004011CB 에 도달했을 때 해당 주소로 넘어가야하는데 다시 0x0040108C 여기에 가는 걸 알 수 있다. 즉, 004011CB 를 부르는 함수 쪽이 이상하다는 걸 알 수 있고 이 함수를 분석해야한다.

004011CB	55	PUSH EBP	
004011CC	8BEC	MOV EBP,ESP	
004011CE	81C4 D4FEFF	ADD ESP,-12C	
004011D4	53	PUSH EBX	
004011D5	56	PUSH ESI	
004011D6	57	PUSH EDI	
004011D7	6A 00	PUSH 0	
004011D9	6A 0F	PUSH 0F	
004011DB	E8 3E020000	CALL <JMP.&kernel32.CreateToolhelp32Snapshot>	ProcessID = 0 Flags = TH32CS_SNAPALL CreateToolhelp32Snapshot
004011DE	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	
004011E3	8DB5 D4FEFF	LEA ESI,DWORD PTR SS:[EBP-12C]	
004011E9	8D3D 4C304000	LEA EDI,DWORD PTR DS:[40304C]	
004011EF	56	PUSH ESI	
004011F0	FF75 FC	PUSH DWORD PTR SS:[EBP-4]	pProcessentry hSnapshot
004011F3	E8 38020000	CALL <JMP.&kernel32.Process32First>	Process32First
004011F8	85C0	TEST EAX,EAX	
004011FA	74 2B	JE SHORT Debugger.00401227	
004011FC	8D46 24	LEA EAX,DWORD PTR DS:[ESI+24]	
004011FF	50	PUSH EAX	String2 String1 => "OLLYDBG.EXE"
00401200	57	PUSH EDI	lstrcmpiA
00401201	E8 3C020000	CALL <JMP.&kernel32.lstrcmpiA>	
00401206	85C0	TEST EAX,EAX	
00401208	74 2A	JE SHORT Debugger.00401234	
0040120A	56	PUSH ESI	pProcessentry hSnapshot
0040120B	FF75 FC	PUSH DWORD PTR SS:[EBP-4]	Process32Next
0040120E	E8 23020000	CALL <JMP.&kernel32.Process32Next>	
00401213	85C0	TEST EAX,EAX	

'F7' 이용해서 들어가보면 여러가지 msdn 함수가 나온다.

#### \* msdn 함수 설명

CreateToolhelp32snapshot : 지정된 프로세스의 스냅샷 이러한 프로세스에서 사용되는 힙, 모듈 및 스레드를 사용

반환 값 -> 함수가 성공하면 지정된 스냅샷 열린 핸들을 반환

Process32First : 시스템 스냅샷 발생한 첫 번째 프로세스에 대한 정보를 검색

반환 값 -> 프로세스 목록의 첫 번째 항목이 버퍼에 복사되었으면 TRUE(1) 를 반환

lstrcmpiA : 두 문자 문자열을 비교

반환 값 -> lpString1 가리키는 문자열이 lpString2 가리키는 문자열보다 작으면 반환 값은 음수 아니면 양수

Process32Next : 시스템 스냅샷 기록된 다음 프로세스에 대한 정보를 검색

반환 값 -> 프로세스 목록의 첫 번째 항목이 버퍼에 복사되었으면 TRUE(1) 를 반환

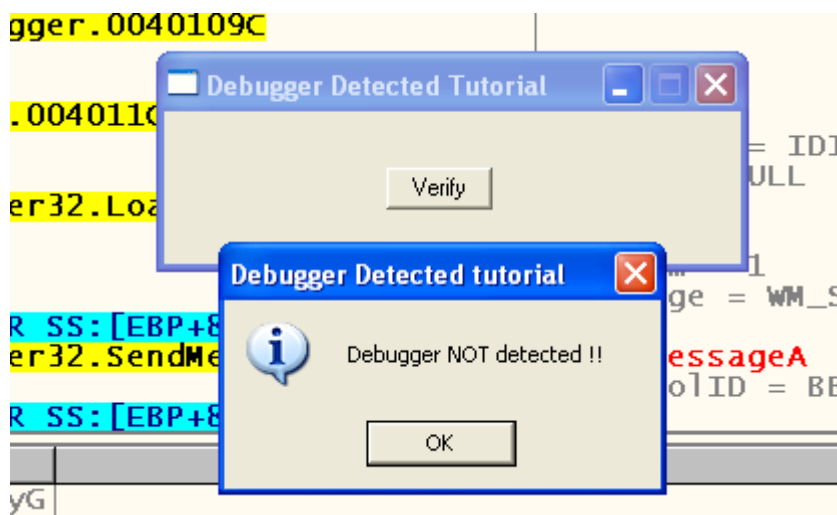
004011FF	50	PUSH EAX	String2 = "[System Process]"
00401200	57	PUSH EDI	String1 => "OLLYDBG.EXE"
00401201	E8 3C020000	CALL <JMP.&kernel32.lstrcmpiA>	lstrcmpiA
00401206	85C0	TEST EAX,EAX	
00401208	74 2A	JE SHORT Debugger.00401234	
0040120A	56	PUSH ESI	pProcessentry
0040120B	FF75 FC	PUSH DWORD PTR SS:[EBP-4]	hSnapshot
0040120E	E8 23020000	CALL <JMP.&kernel32.Process32Next>	Process32Next
00401213	85C0	TEST EAX,EAX	
00401215	74 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	74 0F	JE SHORT Debugger.00401234	
00401225	EB F3	JMP SHORT Debugger.0040120A	

lstrcmpiA 에서 프로세스 문자열을 비교하는 걸 볼 수 있는데 'OLLYDBG.EXE' 올리디버그를 찾는 걸 알 수 있다. (그래서 올리디버그에서 실행하면 다른 창이 나옴)

lstrcmpiA 함수는 EAX = 1 을 반환하고 루프를 도는 걸 알 수 있다. 아마 현재 실행 중인 프로세스 중 OLLYDBG.EXE 가 나올 때까지 해당 루프를 도는 거 같다.

00401208	74 2A	JE SHORT Debugger.00401234	
0040120A	56	PUSH ESI	
0040120B	FF75 FC	PUSH DWORD PTR SS:[EBP-4]	pProcessentry
0040120E	E8 23020000	CALL <JMP.&kernel32.Process32Next>	hSnapshot
00401213	85C0	TEST EAX,EAX	Process32Next
00401215	74 10	JE SHORT Debugger.00401227	
00401217	8D46 24	LEA EAX,DWORD PTR DS:[ESI+24]	String2
0040121A	50	PUSH EAX	String1
0040121B	57	PUSH EDI	lstrcmpiA
0040121C	E8 21020000	CALL <JMP.&kernel32.lstrcmpiA>	
00401221	85C0	TEST EAX,EAX	
00401223	74 0F	JE SHORT Debugger.00401234	
00401225	EB E3	JMP SHORT Debugger.0040120A	
00401227	FF75 FC	PUSH DWORD PTR SS:[EBP-4]	hObject
0040122A	E8 E9010000	CALL <JMP.&kernel32.CloseHandle>	CloseHandle
0040122F	5F	POP EDI	
00401230	5E	POP ESI	
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

OLLYDBG.EXE 를 찾게 되면 0x00401223 에서 점프하는 걸 알 수 있고 그럼 디버거 사용하고 있다는 창이 나오게 되니까 점프를 하면 안된다.



안넘어가게 하기 위해서는 JE 00401234 -> NOP 으로 바꾸고 저장해서 올리디버그로 실행해보면 해당 창이 나오는 걸 알 수 있다.

문제 해결!