악성코드 분석 보고서

(sand-reversingwithlena-tutorials)

2025.05.27

01

```
RsrcName = 4.
hInst => 00400000
LoadIconA

RsrcName = IDC_ARROW
hInst = NULL
LoadCursorA

hTemplateFile = NULL
Attributes = READONLY|HIDDEN|SYSTEM|ARCHIVE|TEMPORARY|402048
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Text = "Evaluation period out of date. Purchase new license"
hOwner = NULL
MessageBoxA
ExitProcess

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

```
Style = MB_OK|MB_APPLMODAL
Title = " Key File ReverseMe"
Text = "You really did it! Congratz !!!"
hOwner = NULL
MessageBoxA
ExitProcess
```

```
Style = MB_OK|MB_APPLMODAL
Title = " Key File ReverseMe"
Text = "Keyfile is not valid. Sorry."
hOwner = NULL
MessageBoxA
ExitProcess
```

LoadIconA : 연결된 실행 파일(.exe) 파일에서 지정된 아이콘 리소스를 로드하는 함수

LoadCursorA : 실행 파일(.EXE) 파일에서 지정된 커서 리소스를 로드하는 함수

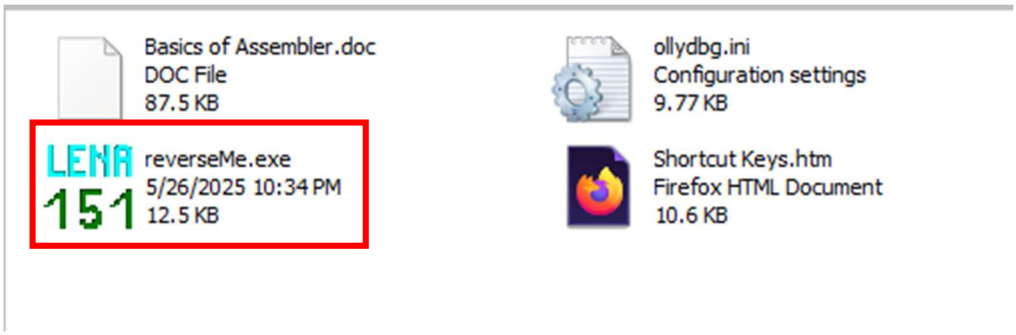
CreateFileA : 파일 또는 I/O 디바이스를 만들거나 여는 함수

MessageBoxA : 시스템 아이콘, 단추 집합 및 상태 또는 오류 정보와 같은 간단한 애플리케이션 관련 메시지가 포함된 모달 대화 상자를 표시하는 함수

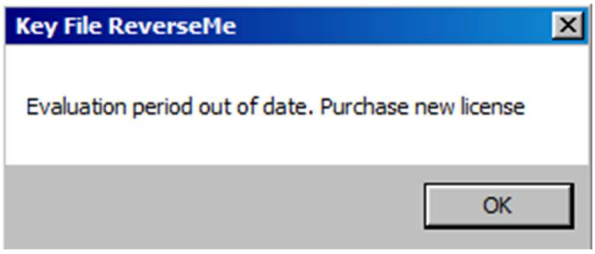
ExitProcess : 호출 프로세스 및 모든 스레드를 종료하는 함수

ReadFile : 지정된 파일 또는 I/O(입출력) 디바이스에서 데이터를 읽는 함수

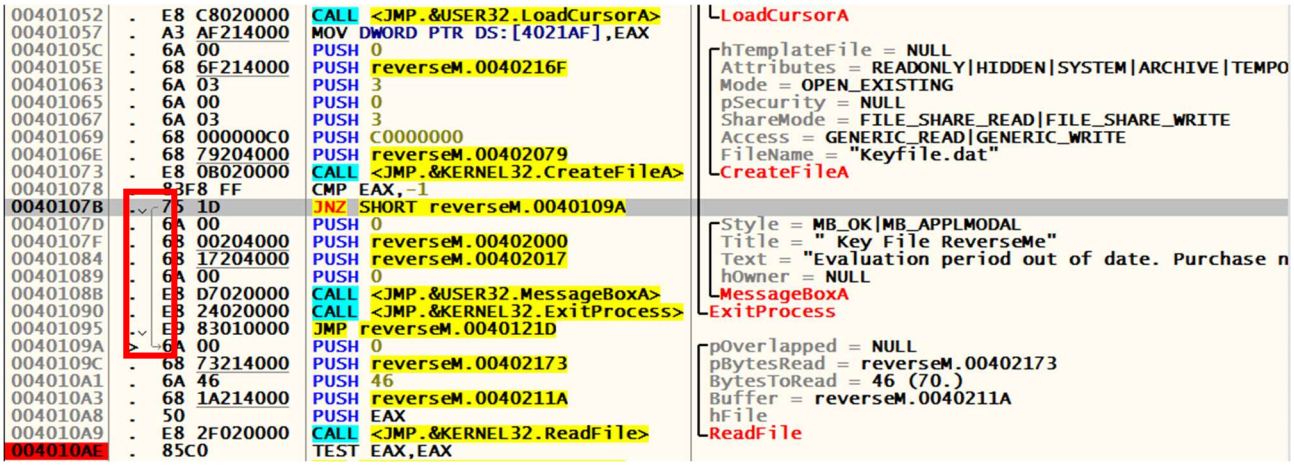
LoadIconA, LoadCursorA 함수는 이걸 분석하는데 중요한 함수가 아닌거 같음



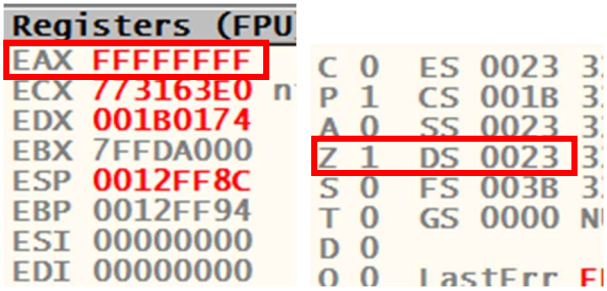
현재 빨간 박스를 클릭할 경우



해당 구문의 메시지 박스를 볼 수 있음



0x00401073에 있는 call 함수가 eax에 FFFFFFFF를 저장하는 걸 볼 수 있음. (FFFFFFFF = -1) 그럼 cmp에서 eax 와 -1를 비교할 경우 둘 다 같으니 ZF = 1을 저장함.



JNZ : 결과가 0이 아닐 때 점프

0x0040107B에서 0이 아닐 때 점프인데 ZF=1 이라서 점프 안하는 걸 볼 수 있음.

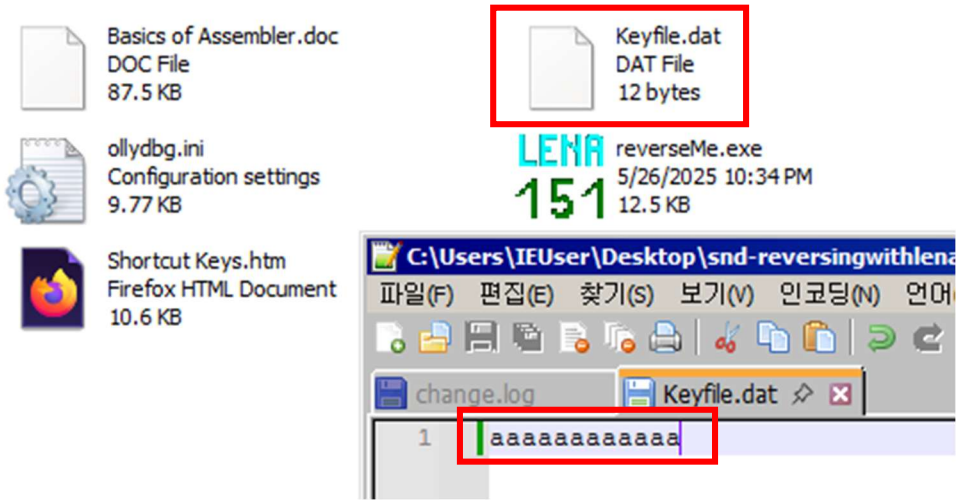
그럼 점프하기 위해서는 ZF=0이 되어야하고 0x00401073에 있는 call함수가 -1을 불러오면 안됨.

```
hTemplateFile = NULL
Attributes = READONLY|HIDDEN|SYSTEM|ARCHIVE|TEMPORARY|402048
Mode = OPEN_EXISTING
pSecurity = NULL
ShareMode = FILE_SHARE_READ|FILE_SHARE_WRITE
Access = GENERIC_READ|GENERIC_WRITE
FileName = "Keyfile.dat"
CreateFileA
```

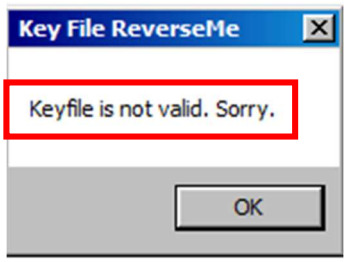
CreateFileA의 반환 값이 -1이 안되기 위해서는 Mode인자를 봐야함.

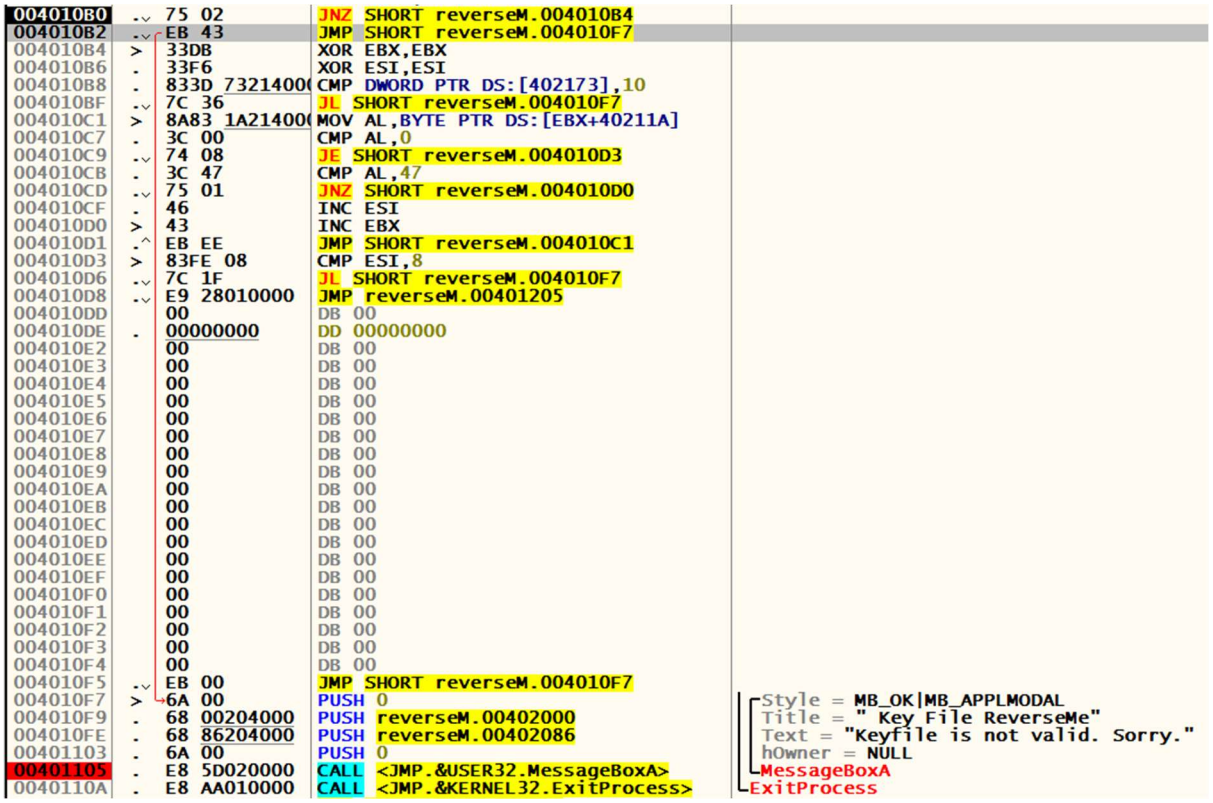
(파일이 있는 경우에만 파일 또는 디바이스를 열고 지정된 파일 또는 디바이스가 없으면 함수가 실패)

즉, 실패하지 않기 위해서는 Keyfile.dat라는 파일이 있어야함.

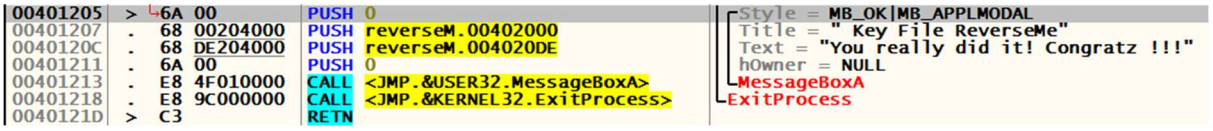


생성하고 파일 안에 아무 문자열이나 집어 넣고 실행할 경우 메시지의 내용이 바뀐 걸 볼 수 있음.

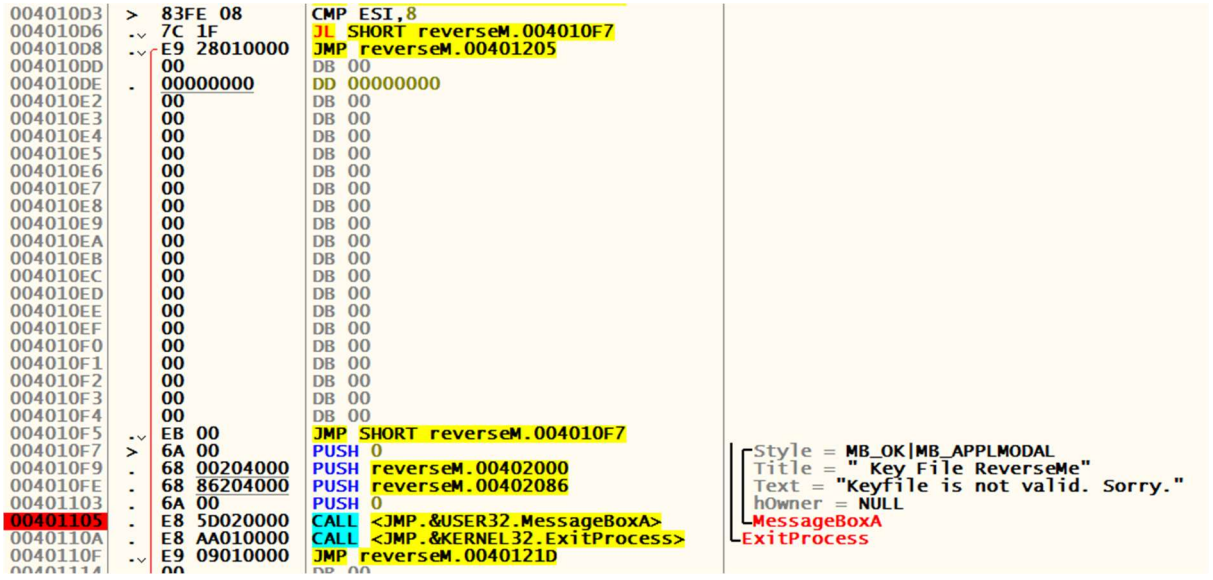




0x004010B2에서 0x004010F7로 점프하는 걸 볼 수 있음.



우리는 여기에 도달해서 저 메시지 박스를 출력해야함. 저기로 점프하는 주소를 찾아야 함.



이걸 보면 0x004010D8에서 점프하는 걸 볼 수 있음. 여기까지 도달하기 위해서는

```
004010A9  .   E8 2F020000   CALL <JMP.&KERNEL32.ReadFile>
004010AE  .   85C0          TEST EAX,EAX
004010B0  .v  75 02         JNZ SHORT reverseM.004010B4
004010B2  .v  EB 43         JMP SHORT reverseM.004010F7
004010B4  >   33DB          XOR EBX,EBX
```

0x004010B0에서 점프해 0x004010B4로 가야함.

ZF=1이어야함. 즉, TEST EAX, EAX를 (TEST : TEST연산은 오퍼랜드끼리 AND연산) EAX = 1이 나와야함. 그럼 그 전 call 함수에서 eax=1 나와야함. ReadFileA의 반환 값이 1이 되기 위해서는 함수가 성공해야함. (성공했다는 걸 볼 수 있음)

```
004010B4  >   33DB          XOR EBX,EBX
004010B6  .   33F6          XOR ESI,ESI
004010B8  .   833D 7321400( CMP DWORD PTR DS:[402173],10
```

XOR EBX, EBX로 인해서 EBX=00000000, XOR ESI, ESI로 인해서 ESI=00000000로 변경된 걸 볼 수 있음.

```
Registers (FPU)
EAX 00000001
ECX 75D59E17 ke
EDX 77906C74 nt
EBX 00000000
ESP 0012FF8C AS
EBP 0012FF94
ESI 00000000
EDI 00000000
```

```
004010B8  .   833D 7321400( CMP DWORD PTR DS:[402173],10
004010BF  .v  7C 36         JL SHORT reverseM.004010F7
004010C1  >   8A83 1A21400( MOV AL,BYTE PTR DS:[EBX+40211A]
```

지금 그대로 실행해보면 0x004010BF에서 점프해서 0x004010F7로 가는 걸 볼 수 있음.

저기로 점프 안하기 위해서는 (JL: cmp a, b에서 a가 작을 때 점프) DS:[402173]이 10h보다 커야 점프 안하는 걸 볼 수 있음. 즉, 10h -> 16(10진수)이상이어야함.

DS:[402173]을 또 어디서 사용하나 찾아보면 ReadFile함수에서 pBytesToRead에서 사용하는 걸 볼 수 있음.

pBytesToRead : 함수 호출 후 실제 읽혀진 데이터의 크기를 저장할 메모리의 주소 전달

Buffer : 읽어온 내용을 저장할 메모리의 시작 주소를 설정

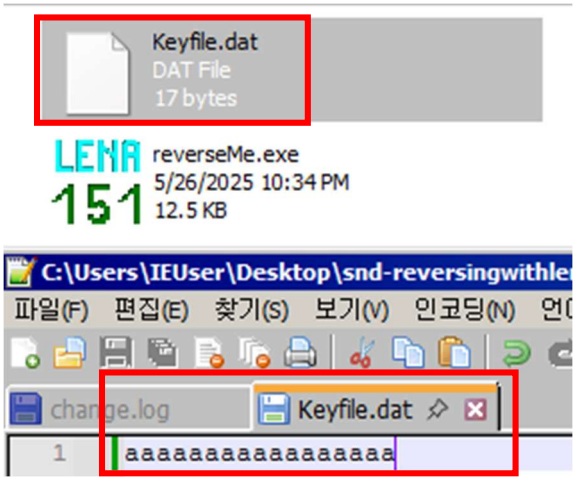
```
00402170  00 00 00 0C
```

위치를 보면 0C로 나와있음 10진수로 고치면 12임.

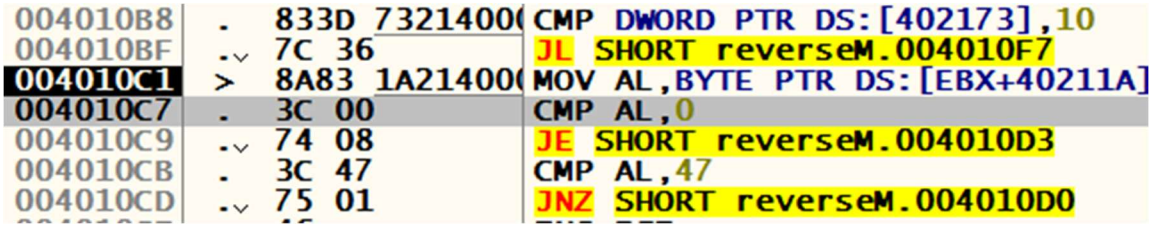
```
00402110  00 00 00 00 00 00 00 00 00 00 61 61 61 61 61 61  ..........aaaaaa
00402120  61 61 61 61 61 61 00 00 00 00 00 00 00 00 00 00  aaaaaa..........
```

Buffer인자에 주소를 보면 이렇게 a가 12개인걸 볼 수 있음.

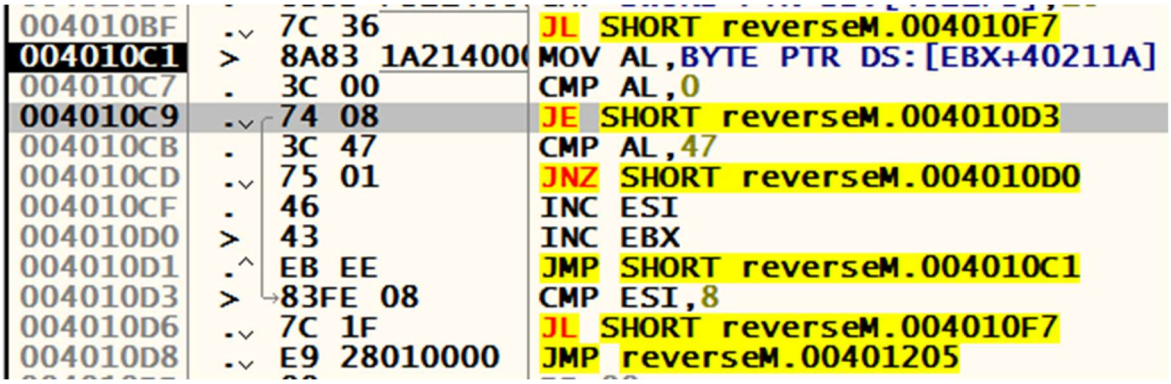
즉, DS:[402173]에 10h(16이상)가 저장되어야 한다는 걸 볼 수 있음.



a를 17개 저장하고 실행해보면



점프 안하고 내려온 걸 볼 수 있음.



0x004010D8로 가기 위해서는 0x004010C9에서 점프해야한다는 걸 볼 수 있음.

(JE: 비교 결과가 같을 때 점프) AL=0 이 되어야하는 걸 볼 수 있음.

하지만 0x40211A로 가서 보면 61이 저장되어 있는 걸 볼 수 있음.

0x004010C9 – 0x004010D3에 있는 코드를 해석해보면

0x004010CD가 ZF=1이어야 점프하는 걸 볼 수 있고 AL이 47h -> 'G'인 걸 볼 수 있고 EBX가 1씩 증가하는 걸 볼 수 있고 0x004010C1로 점프하는 걸 볼 수 있음

근데 DS:[EBX+0x40211A]인걸 볼 수 있고 Keyfile.dat에 있는 내용을 하나씩 읽어오고 다 읽을 때까지 0x004010C1로 점프할거라는 걸 알 수 있음.

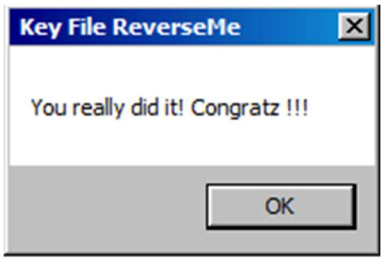
그럼 파일의 내용을 다 읽은 후에 DS:[EBX+0x40211A]의 값은 0이 되어 AL=0이 될거고

0x004010C9에서 점프가 되어 0x004010D3으로 가는 걸 볼 수 있음.

근데 0x004010D6 JL에서 점프 안하긴 위해서는 CMP ESI, 8을 봐야하는데 ESI가 8보다 커야한다는걸 볼 수 있음.

즉, 0x004010C9 – 0x004010D3여기의 내용을 정리해보면 Keyfile.dat내용이 'G'가 8개 이상은 꼭 있어야하고 총 길이가 16개 이상 있어야한다는 걸 알 수 있음.

이렇게 저장하고 실행해보면



이 메시지 창이 뜨는 걸 볼 수 있음.

해결 완료!