

리버스 엔지니어링 보고서

write by 허00

abex level1 분석&풀이

목차

1.서문

- 1-1 개요
- 1-2 목표
- 1-3 목적
- 1-4 사용한 분석툴

2.정적 분석

- 2-1 외관상 특징
 - 2-1-1 아이콘, 확장자
- 2-2 import된 항목
- 2-3 export된 항목
- 2-4 문자열
- 2-5 PE구조상 특이사항

3.동적 분석

- 3-1 패킹여부
- 3-2 코드 분석
 - 3-2-1 결정적 단서
 - 3-2-2 역컴파일 결과
 - 3-2-3 문제 해결 방법&풀이
- 3-3 포트 분석
- 3-4 프로세스 분석
- 3-5 생성파일
- 3-6 레지스트리

4.마치며

- 4-1 발견한 노하우
- 4-2 참신했던 부분
- 4-3 연구, 필요 했던 부분

1.서문

1-1 개요

"abex crackme level 1"을 분석하게 되었다. 분석 후 보고서 작성은 처음이고, 문제 풀이 이상으로 접근한 적도 이번이 처음이다. pe구조 분석, 역컴파일, 코드 분석으로 내 칼을 갈고 닦을 수 있으면 좋겠다.

1-2 목표

"abex crackme level 1"을 단순히 풀이하는 대신 철저하게 분석할 수 있다.

1-3 목적

단순 풀이보다는 철저한 분석과 공부를 겸해서 실전감각을 늘릴 수 있다.

1-4 사용한 분석툴

- peview : pe구조를 한 눈에 보여주는 프로그램이다.
- ollydbg : 프로그램의 흐름, 데이터, 스택등을 한눈에 볼 수 있다.

2.정적 분석

2-1 외관상 특징

2-1-1 아이콘, 확장자

아이콘 : NONE

확장자 : exe(PE)

2-2 import된 항목

pFile	Data	Description	Value
00000A50	0000307C	Hint/Name RVA	0000 GetDriveTypeA
00000A54	0000308C	Hint/Name RVA	0000 ExitProcess
00000A58	00000000	End of Imports	KERNEL32.dll
00000A5C	0000309A	Hint/Name RVA	0000 MessageBoxA
00000A60	00000000	End of Imports	USER32.dll

GetDriveTypeA, ExitProcess, MessageBoxA 함수를 임포트 한다.

GetDriveTypeA 함수 사용으로 볼 때 드라이브 정보에 문제풀이에 대한 힌트가 존재할 것이다.

2-3 export된 항목

NONE

2-4 문자열

00000800	61 62 65 78 27 20 31 73	74 20 63 72 61 63 6B 6D	abex' 1st crackm
00000810	65 00 6D 61 6B 65 20 6D	65 20 74 68 69 6E 6B 20	e.Make me think
00000820	79 6F 75 7D 22 20 48 44	69 73 20 61 20 63 44 2D	your HD is a CD-
00000830	52 6F 6D 2E 00 45 72 72	6F 72 00 4E 61 68 2E 2E	Rom..Error.Nah..
00000840	2E 20 54 68 69 73 20 69	73 20 6E 6F 74 20 61 20	. This is not a
00000850	43 44 2D 52 4F 4D 20 44	72 69 76 65 21 00 59 45	CD-ROM Drive!.YE
00000860	41 48 21 00 4F 6B 2C 20	49 20 72 65 61 6C 6C 79	AH!.Ok, I really
00000870	70 74 68 69 6E 6B 20 74	68 61 74 20 79 6F 75 72	think that your
00000880	20 48 44 20 69 73 20 61	20 43 44 2D 52 4F 4D 21	HD is a CD-ROM!
00000890	20 3A 70 00 63 3A 5C 00	00 00 00 00 00 00 00 00	p.c:\.....

프로그램 내부에 들어 있는 문자열 들이다. 문제에 풀이 조건에 필요한 단
서들이 보인다.

2-5 PE구조상 특이사항

3.동적 분석

3-1 패킹여부

NONE

3-2 코드 분석

```

00401000  6A 00      PUSH 0
00401001  68 00204000  PUSH OFFSET 00402000
00401002  68 12204000  PUSH OFFSET 00402012
00401003  6A 00      PUSH 0
00401004  49 4E000000  CALL <JMP.>USER32.MessageBoxA
00401010  68 32404000  PUSH OFFSET 00402094
00401011  49 4E000000  CALL <JMP.>KERNEL32.GetDriveTypeA
00401012  46         INCESI
00401013  48         DEC ERX
00401014  EB 00      JMP SHORT 00401021
00401021  46         INCESI
00401022  48         DEC ERX
00401023  48         DEC ERX
00401024  74 15      JE SHORT 00401030
00401025  6A 00      PUSH 0
00401026  68 35204000  PUSH OFFSET 00402035
00401027  68 32404000  PUSH OFFSET 00402038
00401028  6A 00      PUSH 0
00401029  49 4E000000  CALL <JMP.>USER32.MessageBoxA
00401030  4B 19      JMP SHORT 00401050
00401031  6A 00      PUSH 0
00401032  68 5E204000  PUSH OFFSET 0040205E
00401033  68 42404000  PUSH OFFSET 00402064
00401034  6A 00      PUSH 0
00401035  49 4E000000  CALL <JMP.>USER32.MessageBoxA
00401036  4B 19      JMP SHORT 00401050
00401037  6A 00      PUSH 0
00401038  68 5E204000  PUSH OFFSET 0040205E
00401039  68 42404000  PUSH OFFSET 00402064
00401040  6A 00      PUSH 0
00401041  49 4E000000  CALL <JMP.>USER32.MessageBoxA
00401042  4B 19      JMP SHORT 00401050
00401043  6A 00      PUSH 0
00401044  68 5E204000  PUSH OFFSET 0040205E
00401045  68 42404000  PUSH OFFSET 00402064
00401046  6A 00      PUSH 0
00401047  49 4E000000  CALL <JMP.>USER32.MessageBoxA
00401048  4B 19      JMP SHORT 00401050
00401049  6A 00      PUSH 0
00401050  68 5E204000  PUSH OFFSET 0040205E
00401051  68 42404000  PUSH OFFSET 00402064
00401052  6A 00      PUSH 0
00401053  49 4E000000  CALL <JMP.>USER32.MessageBoxA
00401054  4B 19      JMP SHORT 00401050
00401055  55 58304000  JMP DWORD PTR DS:[<KERNEL32.GetDriveTypeA>]
00401056  55 58304000  JMP DWORD PTR DS:[<KERNEL32.ExitProcess>]
00401057  55 58304000  JMP DWORD PTR DS:[<USER32.MessageBoxA>]

```

전체코드의 모습 코드크기가 작아서 한 번에 넣을 수 있었다.

3-2-1 결정적 증거

(그림 1)

00401000	6A 00	PUSH 0	Type = MB_OK MB_DEFBUTTON1 MB_APPLMODAL
00401002	68 00204000	PUSH OFFSET 00402000	Caption = "abex" 1st crackme"
00401004	68 12040000	PUSH OFFSET 00402012	Text = "Make me think your HD is a CD-Ron."
0040100C	6A 00	PUSH 0	hOwner = NULL
0040100E	68 4E000000	CALL <JMP.&USER32.MessageBoxA>	USER32.MessageBoxA
00401013	68 94204000	PUSH OFFSET 00402094	RootPath = "c:\\"
00401018	68 38000000	CALL <JMP.&KERNEL32.GetDriveTypeA>	KERNEL32.GetDriveTypeA

“사진1”에서 “make me think your HD is a CD-ROM” 이라고 풀이 방법을 안내하고 있다.

그 후에 `c:\w`를 `GetDriveTypeA` 함수에 전달하고 있다.

(그림 2)

Return code/value	Description
DRIVE_FIXED 3	The drive has fixed media; for example, a hard disk drive or flash drive.
DRIVE_REMOTE 4	The drive is a remote (network) drive.
DRIVE_CDROM 5	The drive is a CD-ROM drive.

GetDriveTypeA의 리턴값에 대한 msdn의 정보이다. 하드디스크, 플래시 드라이브에서는 3을 리턴, cd-rom의 경우에는 5를 리턴한다.

(그림 3)

```
00401010 | . 46      INC ESI
00401011 | . 48      DEC EAX
00401012 | < EB 00   JMP SHORT 00401021
00401021 | > 46      INC ESI
00401022 | . 46      INC ESI
00401023 | . 48      DEC EAX
00401024 | . 3BC6    CMP EAX,ESI
00401026 | < 74 15   JE SHORT 0040103D
00401028 | . 6A 00   PUSH 0
00401029 | . 68 35204000 PUSH OFFSET 00402035
0040102F | . 68 38204000 PUSH OFFSET 00402038
00401034 | . 6A 00   PUSH 0
00401036 | . E8 26000000 CALL <JMP.&USER32.MessageBoxA>
00401038 | < EB 13   JMP SHORT 00401050
0040103D | > 6A 00   PUSH 0
0040103F | . 68 5E204000 PUSH OFFSET 0040205E
00401044 | . 68 64204000 PUSH OFFSET 00402064
00401049 | . 6A 00   PUSH 0
0040104B | . E8 11000000 CALL <JMP.&USER32.MessageBoxA>
00401050 | < EB 06   JMP SHORT 00401058
00401058 | . E8 06000000 CALL <JMP.&KERNEL32.ExitProcess>
```

```
Type = MB_OK;MB_DEFBUTTON1;MB_APPLMODAL
Caption = "Error"
Text = "Nah... This is not a CD-ROM Drive!"
hOwner = NULL
USER32.MessageBoxA

Type = MB_OK;MB_DEFBUTTON1;MB_APPLMODAL
Caption = "YEAH!"
Text = "Ok, I really think that your HD is a CD-ROM! :p"
hOwner = NULL
USER32.MessageBoxA
KERNEL32.ExitProcess
```

(그림 1)에서 호출한 GetDriveTypeA 함수의 리턴값을 조작하는 로직, 그 후 어떤 메시지를 출력할 것인지 결정하는 로직이 붙어 있다. 0040101D부터 00401023 까지는 esi레지스터가 3으로 설정되고, eax 레지스터는 "리턴값-2"가 되고 이를 비교한다. 만일 리턴값이 5 (CD-ROM)이라면 esi와 eax의 값이 같게 되고, 00401026 "JE SHORT 0040103D"의 조건에 따라서 0040103D로 이동할 것이다.

3-2-2 역컴파일 결과

```
level1.c
1  #include<stdio.h>
2  #include<windows.h>
3
4
5
6  int main()
7  {
8      int return_value , esi=0;
9      MessageBox(0,"go","abex 1",0);
10     return_value = GetDriveTypeA("c:\\");
11     esi++;
12     return_value--;
13     esi+=2;
14     return_value--;
15
16     if(return_value == esi )
17         MessageBox(0,"y","abex 1",0);
18     else
19         MessageBox(0,"n","abex 1",0);
20
21 }
```

3-2-3 문제 해결방법 & 풀이

Sol1. 리턴값을 중간에 조작하는 "dec eax"를 "nop"로 바꾼다.

• 46	INC ESI
• 90	NOP
• EB 00	JMP SHORT 00401021
> 46	INC ESI
• 46	INC ESI
• 90	NOP
• 3BC6	CMP EAX,ESI

Sol2. "JE SHORT 0040103D"를 "JNE SHORT 0040103D" 로 바꾼다.

EB 00	JMP SHORT 00401021
46	INC ESI
46	INC ESI
48	DEC EAX
3BC6	CMP EAX,ESI
75 15	JNE SHORT 0040103D

아래와 같이 성공 메시지가 출력된다.



3-3 포트 분석

NONE

3-4 프로세스 분석

NONE

3-5 생성파일

NONE

3-6 레지스트리

NONE

4.마치며

4-1 발견한 노하우

1. 코드 내에서 함수 호출방법

(그림4)

00401036	• E8 26000000	CALL <JMP.&USER32.MessageBoxA>
0040103B	• EB 13	JMP SHORT 00401050
0040103D	> 6A 00	PUSH 0
0040103F	• 68 2E204000	PUSH OFFSET 0040205E
00401044	• 60 24204000	PUSH OFFSET 00402054
00401049	• 6A 00	PUSH 0
0040104B	• E8 11000000	CALL <JMP.&USER32.MessageBoxA>
00401050	> E8 06000000	CALL <JMP.&KERNEL32.ExitProcess>
00401055	• FF25 00000000	JMP DWORD PTR DS:[<&KERNEL32.SetDxLiveType>]
0040105B	• FF25 54304000	JMP DWORD PTR DS:[<&KERNEL32.ExitProcess>]
00401061	• FF25 5C304000	JMP DWORD PTR DS:[<&USER32.MessageBoxA>]

분석중에 코드영역의 00401055부터 "ff25" 패턴이 반복되어서 나타나고 있는 것을 발견 하였다. 추가적으로 "2-2 import 된 항목"의 함수 이름들이 보인다. 조사 결과 IAT(Import Address Table)에 들어 있는 함수주소를 호출하기 위해서 사용된다고 한다.

한편 코드내에 "CALL <xxx>" 구문으로 함수를 호출하는 것을 볼 수 있다. 00401036의 MessageBoxA를 호출하는 방법을 분석해보았다. "E8+26000000" 라

는 값이 들어 있는데 여기서 주소를 계산할 때에는 리틀엔디안 방법을 이용해야 한다. 즉 26000000은 00000026인 셈이다.

한편 00401061이 "JMP DWORD PTR DS:[<&USER32.MessageBoxA>]"로 MessageBoxA를 가리키고 있는 것을 볼 수 있다. 00000026이 가리키는 곳은 상대주소이다. 즉 $\{00401036 + (E8\ 26\ 00\ 00\ 00)5\text{byte}\} + 00000026 = 00401061$ 이라는 계산이 나온다. 정리하면 (그림5)와 같다.
(그림5)



4-2 참신했던 부분

리버싱 문제에 대한 풀이 방법이 여러 가지가 있다는 것이 신기했다.

4-3 연구, 필요 했던 부분

1. GetDriveTypeA 함수
2. 올리디버거 사용법
3. 리틀엔디안