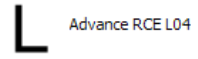
Advance RCE L04

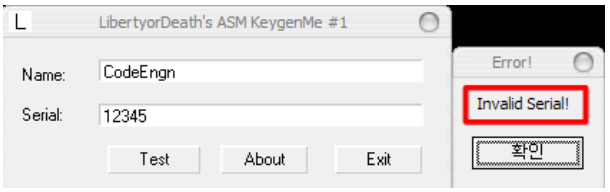
2010년 9월 21일 화요일

오전 1:22

파일 확인

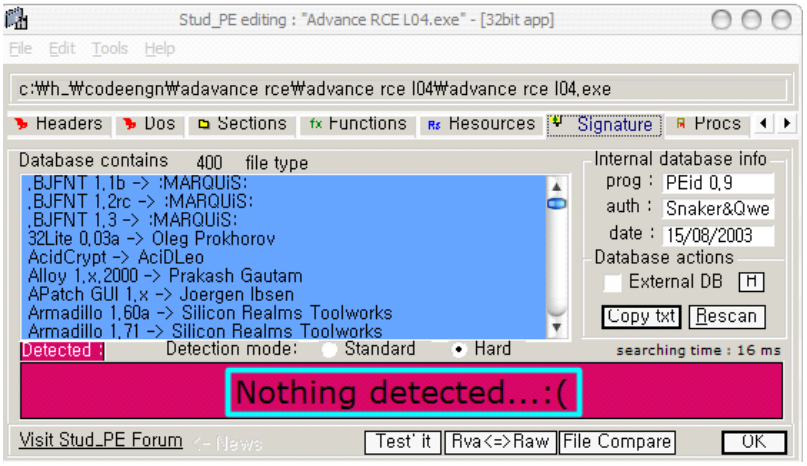


프로그램 실행



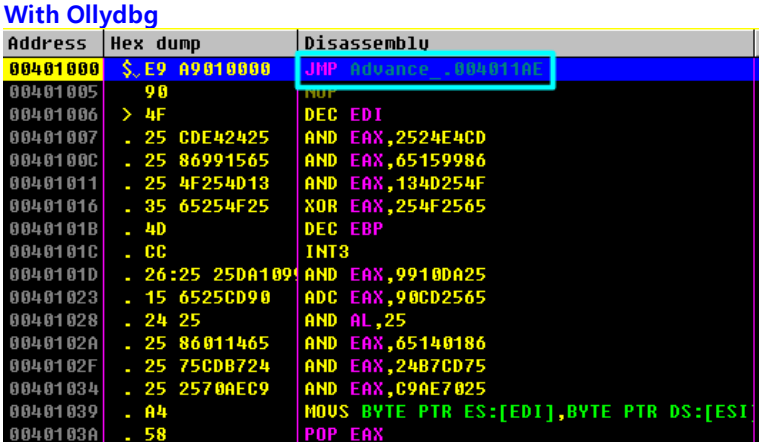
2개의 TextBox 에 값을 입력 받아서 name 에 따른 Serial 을 생성하여 검사하는 프로그램이다

With Stud_PE



알려진 Unpacking 기법 또는 Signature 가 아니므로 확인을 직접 해보아야 한다.

With Ollydbg



Code 를 Debugger 를 통해 열어본 결과 시작하자마자 004011AE 로 JMP 를 한다.

JMP 후 Code

```
Address   Hex dump          Disassembly                          Comment
004011AE  > BE 06104000     MOV ESI,Advance_.00401006
004011B3  > 8A06            MOV AL,BYTE PTR DS:[ESI]
004011B5  . 34 25           XOR AL,25
004011B7  . 8806            MOV BYTE PTR DS:[ESI],AL
004011B9  . 46              INC ESI
004011BA  . 81FE A7114000   CMP ESI,Advance_.004011A7
004011C0  .^75 F1           JNZ SHORT Advance_.004011B3
004011C2  .^E9 3FFEFFFF     JMP Advance_.00401006
004011C7    CC              INT3
004011C8  .-FF25 08204000   JMP DWORD PTR DS:[<&kernel32.ExitProces  kernel32.ExitProcess
004011CE  .-FF25 00204000   JMP DWORD PTR DS:[<&kernel32.GetModuleH  kernel32.GetModuleHandleA
004011D4  .-FF25 04204000   JMP DWORD PTR DS:[<&kernel32.lstrcmpA>]  kernel32.lstrcmpA
004011DA  .-FF25 10204000   JMP DWORD PTR DS:[<&user32.wsprintfA>]   user32.wsprintfA
004011E0  .-FF25 14204000   JMP DWORD PTR DS:[<&user32.DialogBoxPar  user32.DialogBoxParamA
004011E6  .-FF25 18204000   JMP DWORD PTR DS:[<&user32.EndDialog>]   user32.EndDialog
004011EC  .-FF25 1C204000   JMP DWORD PTR DS:[<&user32.GetDlgItem>]  user32.GetDlgItem
004011F2  .-FF25 20204000   JMP DWORD PTR DS:[<&user32.GetDlgItemTe  user32.GetDlgItemTextA
004011F8  .-FF25 24204000   JMP DWORD PTR DS:[<&user32.LoadIconA>]   user32.LoadIconA
004011FE  .-FF25 28204000   JMP DWORD PTR DS:[<&user32.MessageBoxA>  user32.MessageBoxA
00401204  .-FF25 2C204000   JMP DWORD PTR DS:[<&user32.SendMessageA  user32.SendMessageA
0040120A  .-FF25 30204000   JMP DWORD PTR DS:[<&user32.SetFocus>]    user32.SetFocus
```

00401006 이 001011A7 이 될때까지 Loop 수행 후 00401006 으로 JMP 하게 된다.

- **이때 00401006 의 주소에 있는 값을 1Byte 씩 읽어 들여 25 와 xor 연산**을 하여 값을 바꾼다.
  - **이로 인해 00401006 의 값을 바꾸어 Code 를 Unpacking** 하는 것을 알 수 있다.

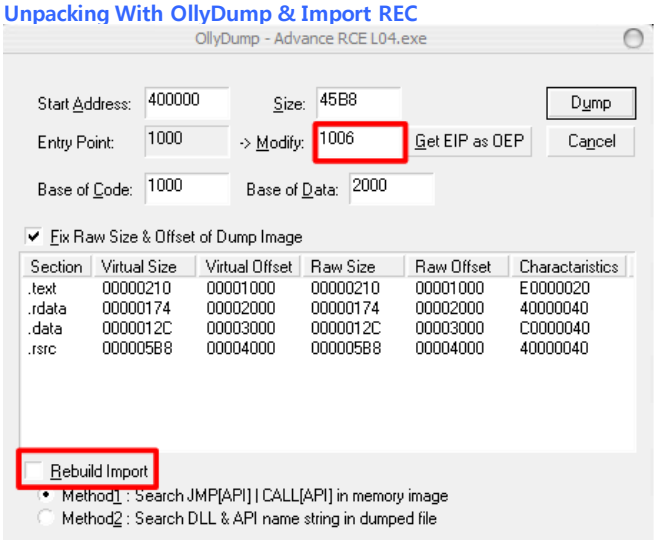
JMP 후 Code

```
Address   Hex dump          Disassembly                          Comment
00401006  > 6A 00           PUSH 0
00401008  ? E8 C1010000     CALL <JMP.&kernel32.GetModuleHandleA>
0040100D  ? A3 BC304000     MOV DWORD PTR DS:[4030BC],EAX
00401012  ? 6A 00           PUSH 0
00401014  ? 68 36104000     PUSH Advance_.00401036
00401019  ? 6A 00           PUSH 0
0040101B  . 68 E9030000     PUSH 3E9
00401020  ? FF35 BC304000   PUSH DWORD PTR DS:[4030BC]
00401026  ? E8 B5010000     CALL <JMP.&user32.DialogBoxParamA>
0040102B  ? A3 24314000     MOV DWORD PTR DS:[403124],EAX
00401030  ? 50              PUSH EAX
00401031  ? E8 92010000     CALL <JMP.&kernel32.ExitProcess>
00401036  ? 55              PUSH EBP
00401037  ? 8BEC            MOV EBP,ESP
00401039  . 817D 0C 10010   CMP DWORD PTR SS:[EBP+C],110
00401040    75              DB 75                                CHAR 'u'
00401041    38              DB 38                                CHAR '8'
00401042    68              DB 68                                CHAR 'h'
```

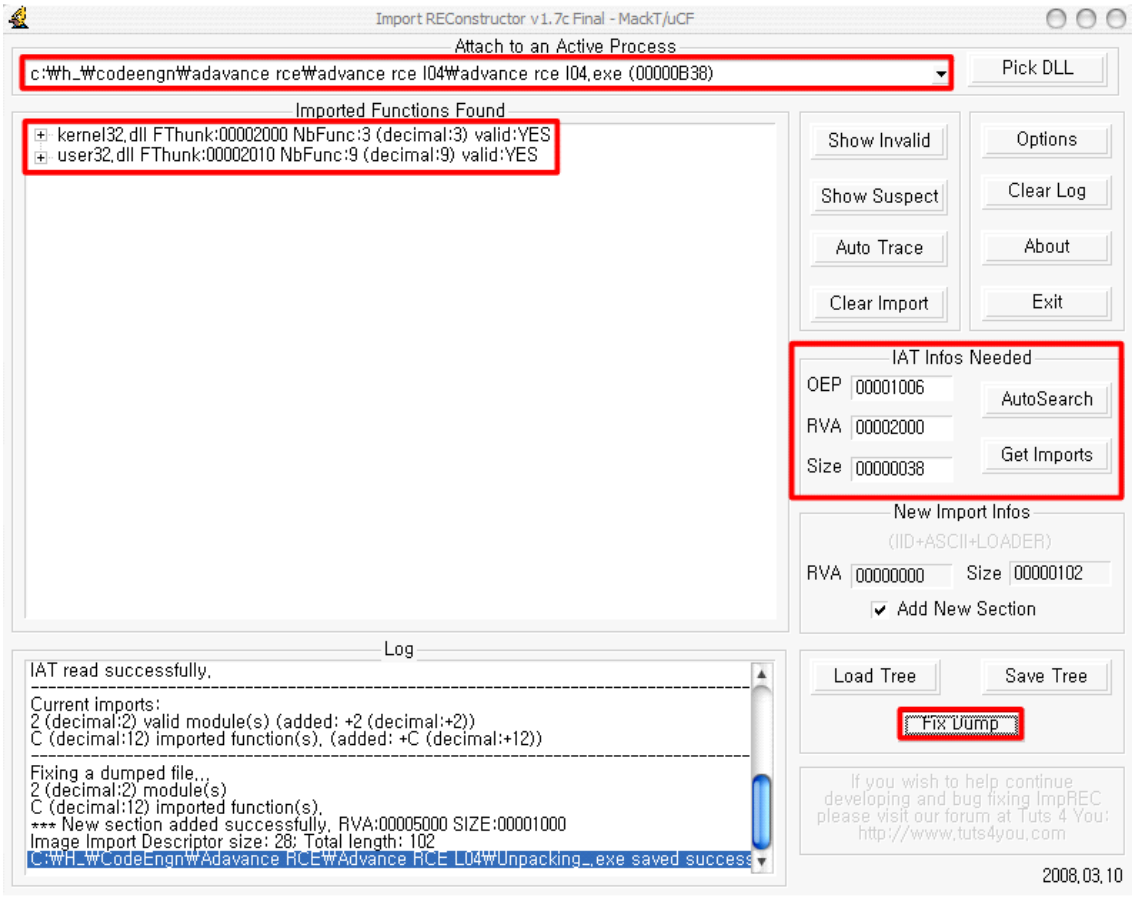
Loop 에 의해 만들어진 Code 가 보이며, 여러 API 함수를 볼 수 있다.

- **이 지점이 OEP 이며 OllyDump Plugin 과 Import REC 로 Unpacking 을 수**행하여 보았다.

Unpacking With OllyDump & Import REC

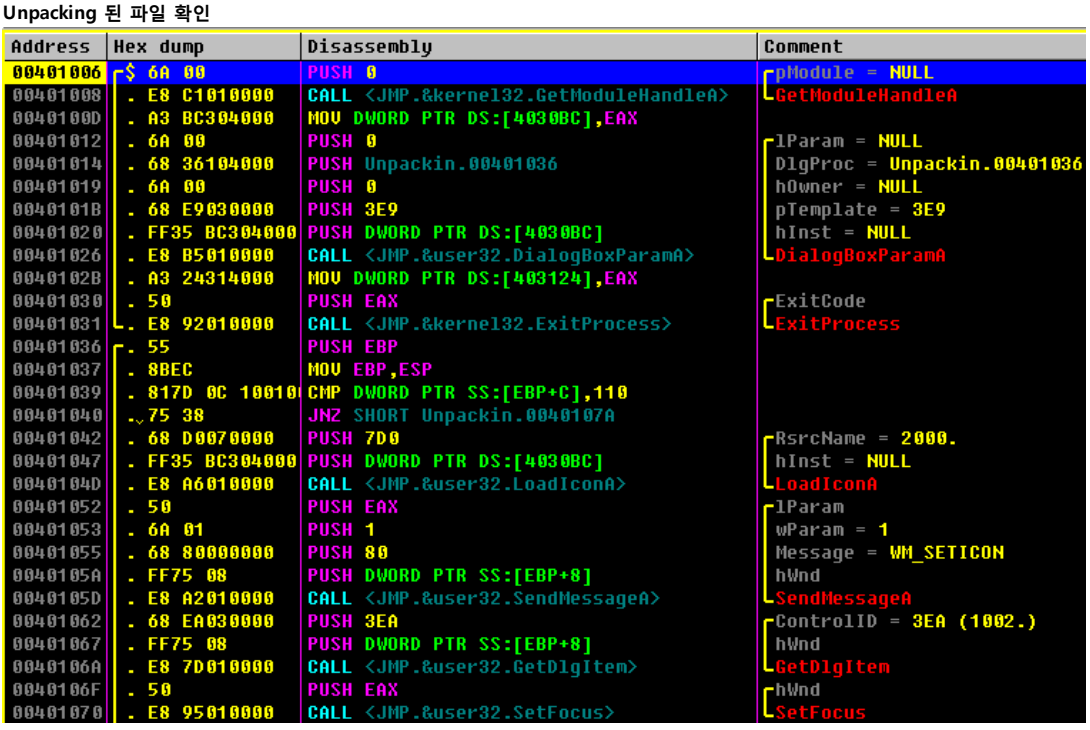


Entry Point 를 1006 으로 수정하고 OllyDump 에서 지원하는 Import 기능은 사용하지 않았다.



Import REC 를 통해 Import 함수를 복원 시켰다.

Unpacking 된 파일 확인

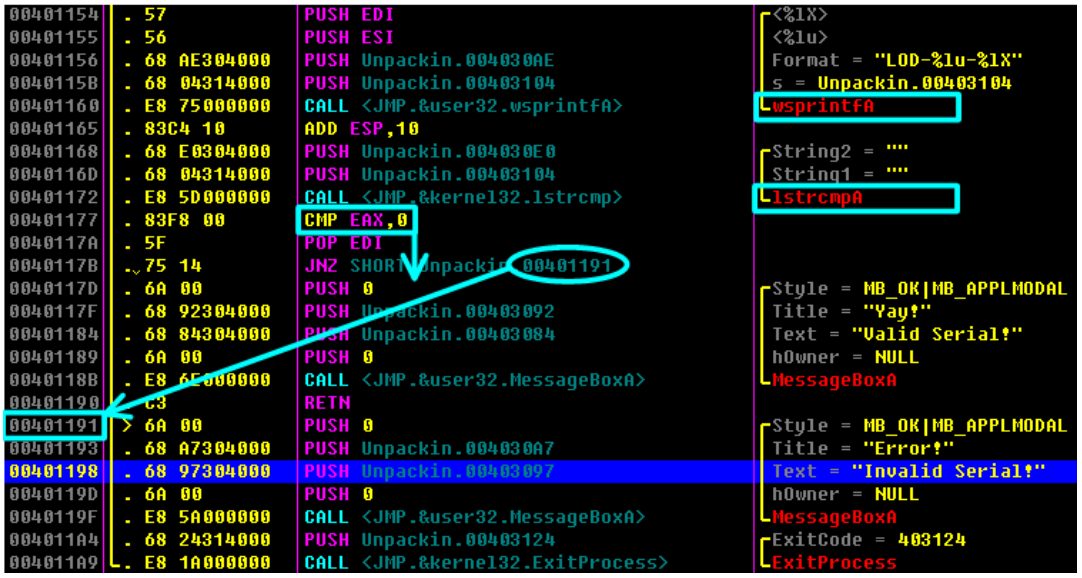


Address	Hex dump	Disassembly	Comment
00401006	$ 6A 00	PUSH 0	pModule = NULL
00401008	. E8 C1010000	CALL <JMP.&kernel32.GetModuleHandleA>	GetModuleHandleA
0040100D	. A3 BC304000	MOV DWORD PTR DS:[4030BC],EAX	
00401012	. 6A 00	PUSH 0	lParam = NULL
00401014	. 68 36104000	PUSH Unpackin.00401036	DlgProc = Unpackin.00401036
00401019	. 6A 00	PUSH 0	hOwner = NULL
0040101B	. 68 E9030000	PUSH 3E9	pTemplate = 3E9
00401020	. FF35 BC304000	PUSH DWORD PTR DS:[4030BC]	hInst = NULL
00401026	. E8 B5010000	CALL <JMP.&user32.DialogBoxParamA>	DialogBoxParamA
0040102B	. A3 24314000	MOV DWORD PTR DS:[403124],EAX	
00401030	. 50	PUSH EAX	ExitCode
00401031	. E8 92010000	CALL <JMP.&kernel32.ExitProcess>	ExitProcess
00401036	. 55	PUSH EBP	
00401037	. 8BEC	MOV EBP,ESP	
00401039	. 817D 0C 10010	CMP DWORD PTR SS:[EBP+C],110	
00401040	. 75 38	JNZ SHORT Unpackin.0040107A	
00401042	. 68 D0070000	PUSH 7D0	RsrcName = 2000.
00401047	. FF35 BC304000	PUSH DWORD PTR DS:[4030BC]	hInst = NULL
0040104D	. E8 A6010000	CALL <JMP.&user32.LoadIconA>	LoadIconA
00401052	. 50	PUSH EAX	lParam
00401053	. 6A 01	PUSH 1	wParam = 1
00401055	. 68 80000000	PUSH 80	Message = WM_SETICON
0040105A	. FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd
0040105D	. E8 A2010000	CALL <JMP.&user32.SendMessageA>	SendMessageA
00401062	. 68 EA030000	PUSH 3EA	ControlID = 3EA (1002.)
00401067	. FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd
0040106A	. E8 7D010000	CALL <JMP.&user32.GetDlgItem>	GetDlgItem
0040106F	. 50	PUSH EAX	hWnd
00401070	. E8 95010000	CALL <JMP.&user32.SetFocus>	SetFocus

정상적인 파일이 확인 가능하다.

문제 해결

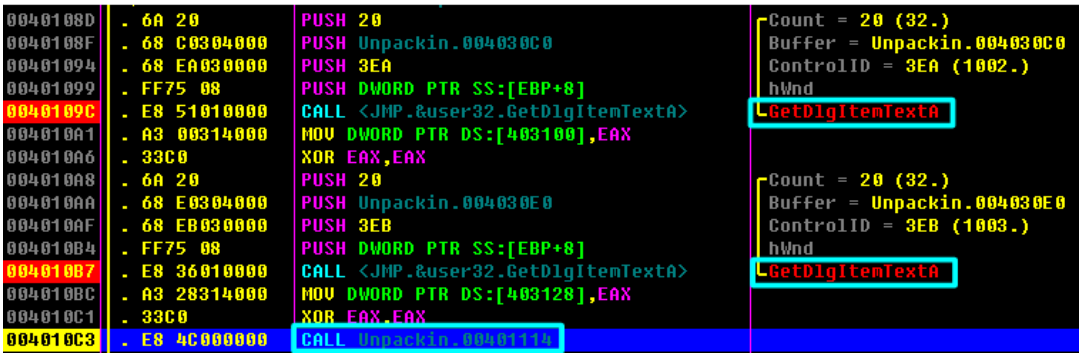
Search For -> All referenced text strings 를 통해 " Invalid Serial! " 이라는 문구를 찾은 후 Code 를 살펴보았다.



Advance RCE L03 문제와 같이 **wsprintfA 함수와 lstrcmpA 함수**가 보인다.

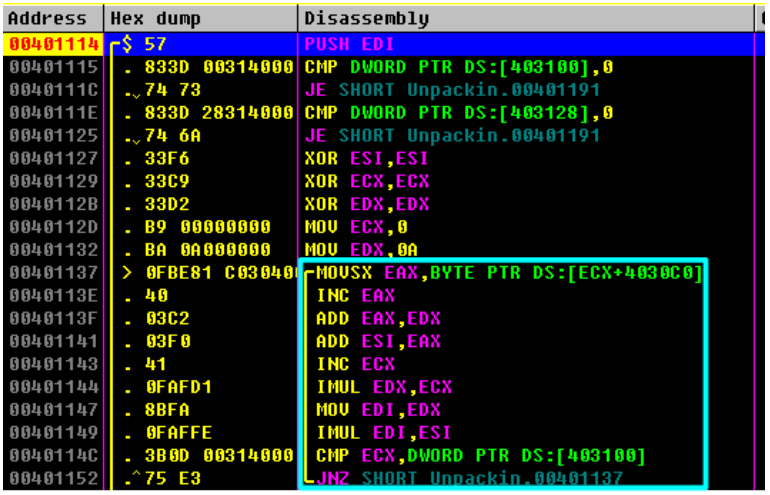
- 이름 값에 대해 **Serial 을 생성한 후, lstrcmpA 함수로 비교하는 것을 예측 가능**하다.

좀더 위로 살펴보면 **GetDlgItemTextA 함수도 보이며 Breakpoint 설정 후 Debugging 을 실행** 시켜 보았다.



Name 과 Serial Text Box 에서 읽어들인 값의 길이를 각각 00403100 과 00403128 에 저장한 후 00401114 함수를 호출한다.

00401114 Code 확인



Loop 문이 보이며, 이를 통해 Serial 값을 생성하는 것을 확인 가능하다.

- **0과 00403100 즉, Name 의 길이 를 비교후 같으면 "Invalid Serial!" 로 JMP**
- **0과 00403128 즉, 사용자가 입력한 Serial 길이 비교 후 같으면 "Invalid Serial!" 로 JMP**
- **ESI, ECX, EDX 를 초기화 후, ECX = 0, EDX = A 를 넣는다.**
- **사용자가 입력한 Name 의 1Byte 를 EAX 에 넣고, EAX 를 1 증가시킨다.**
- **EAX 에 EDX (A) 를 더한 후, 이값을 ESI 에 더한다.**
- **ECX 를 1 증가시킨다. (Loop 수행 조건 위해)**
- **EDX 와 ECX 를 부호있는 곱셈을 수행 후 EDI 에 EDX 를 넣는다.**
- **EDI 와 ESI 를 부호있는 곱셈을 수행 한다.**
- **Name 의 길이와 ECX 비교 후 다시 Loop 를 수행한다.**

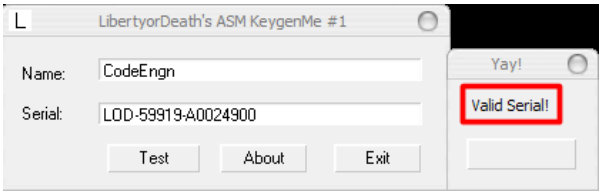
Loop 수행 후 확인

```
00401154  . 57            PUSH EDI                    <%lX>
00401155  . 56            PUSH ESI                    <%lu>
00401156  . 68 AE304000   PUSH Unpackin.004030AE      Format = "LOD-%lu-%lX"
0040115B  . 68 04314000   PUSH Unpackin.00403104      s = Unpackin.00403104
00401160  . E8 75000000   CALL <JMP.&user32.wsprintfA> wsprintfA
00401165  . 83C4 10       ADD ESP,10
00401168  . 68 E0304000   PUSH Unpackin.004030E0      String2 = "12345"
0040116D  . 68 04314000   PUSH Unpackin.00403104      String1 = "LOD-59919-A0024900"
00401172  . E8 5D000000   CALL <JMP.&kernel32.lstrcmp> lstrcmpA
```

wsprintfA 함수에 의해 선언된 Format (LOD-%lu-%lX") 에 맞게 EDI (%lX), ESI (%lu) 를 00403104 에 넣는다.

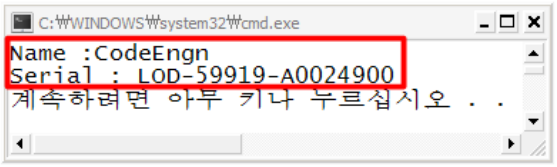
- 그 후 String1 에 00403104 의 값이 사용자가 입력한 Serial 과 비교를 하게 되어 분기 한다.

LOD-59919-A0024900 을 입력 후 결과 확인



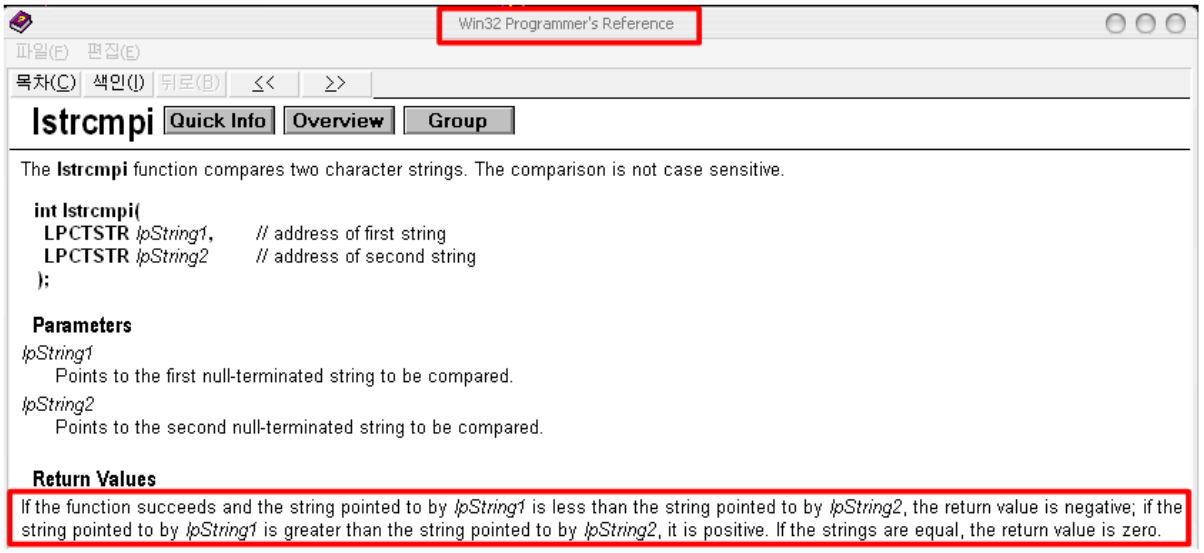
분석한 serial 생성과정을 이용하여 **KeyZen을 생성**해 보았다.

```
#include <stdio.h>
#include <windows.h>

int main()
{
    char name[20];
    int namelen,first,second;

    printf("Name :");
    gets(name);
    namelen = strlen(name);

    __asm
    {
        XOR ESI,ESI
        XOR ECX,ECX
        XOR EDX,EDX
        MOV ECX,0
        MOV EDX,0x0A


    keyloop:
        MOVSX EAX,BYTE PTR DS:[ECX+name]
        INC EAX
        ADD EAX,EDX
        ADD ESI,EAX
        INC ECX
        IMUL EDX,ECX
        MOV EDI,EDX
        IMUL EDI,ESI
        CMP ECX,namelen
        JNZ keyloop

        MOV second,EDI
        MOV first,ESI

    }

    printf("Serial : LOD-%lu-%lX\n",first,second);
    return 0;
}
```



보충 설명

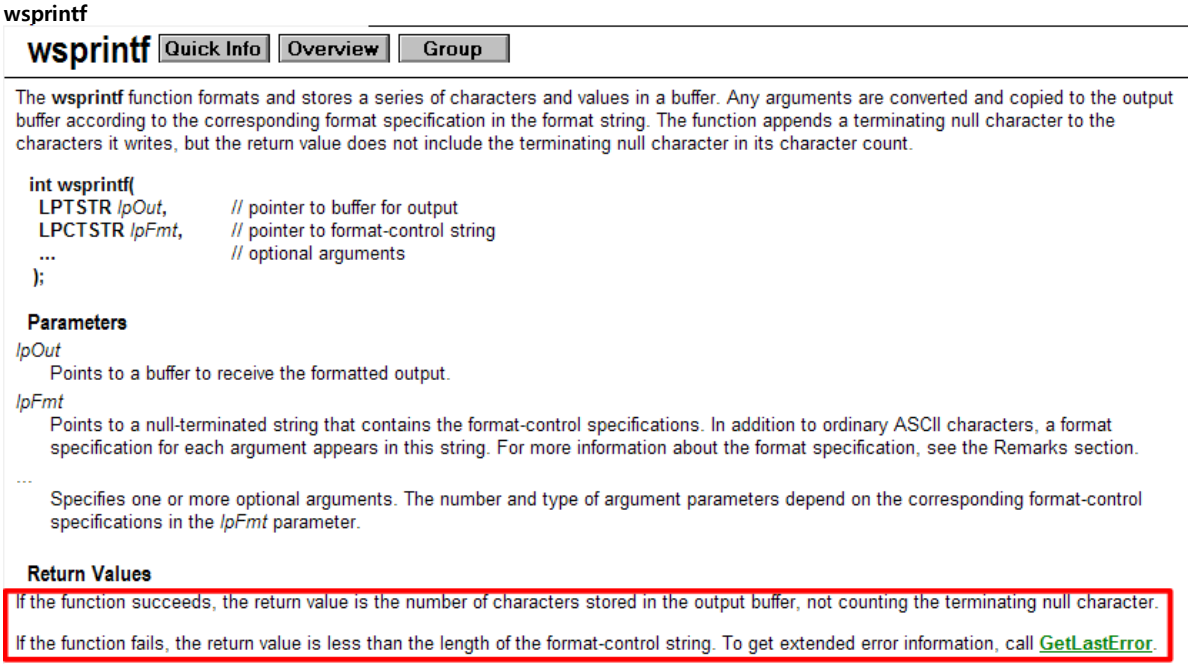
lstrcmpA (Ollydbg 에 Help 에 win32.hlp 를 등록하여 쉽게 확인)



String1 < String2 : -1, String1 = String2 : 0, String1 > String2 : 1 을 Return 한다.

- 함수의 **Return 값은 EAX 레지스터에 반환** 되므로, 위의 프로그램에서 **cmp EAX, 0 을 하여 JNZ 를 이용하여 실패 성공 구문 분기**를 시켰다.

wsprintf



매개 변수 순서에 따라서 위의 코드에서는 s = Out, %u = Fmt, <%u> = arguments 가 된다.

답

LOD-59919-A0024900