Reverse Engineering
Anti-Cracking Techniques

저자 : Nicolaous George (ishtus@astalavista.com) &
Charalambous Glafkos (glafkos@astalavista.com)

편역 : Kancho (kancholove@gmail.com)

이 문서는 www.milw0rm.com의 [papers]에 2008년 6월 26일 게시되었습니다. 문서의 내용을 바탕으로 공부한 내용을 정리하였습니다. 오류나 질문이 있으시면 메일 보내주시기 바랍니다. 역자의 주는 본문 내 *기울임 꼴*이나 각주로 나타내겠습니다.

들어가며

이 문서는 리버스 엔지니어들이 목적을 달성하기 위해 따를 수 있는 대부분의 방법에 대한 이해를 높이기 위한 가이드이다. 또한 시리얼 키 검사나 인증 방법과 같은 소프트웨어의 민감한 정보를 보호하는 더 나은 방법에 대한 조언을 포함하고 있다. 이 문서는 누군가의 이상을 바꾸는 것에 대한 것이 아니라, 리버스 엔지니어링이 더 안전한 세상을 만들 수 있다고 믿는 사람들을 위한 것이다. 만약 당신이 이런 생각을 하는 사람이 아니라면, 이 문서는 당신을 위한 것이 아니다.
이 문서는 리버스 엔지니어들에 의해 사용되는 다양한 기술을 다루고 있지 않으므로 만약 빠진 것이 있다면 망설이지 말고 메일을 보내달라.

할 것들

다룰 주제들:

- PE packer와 암호화 툴
- 온라인 검사
- 악성 코드 분석
- x64 리버스 엔지니어링
- 취약점 발견과 exploiting

다른 주제에 대한 제안도 환영한다. 제안하거나 도움될 것이 있다면 메일 보내달라.

리버스 엔지니어링 툴

인터넷 상에서 많은 리버스 엔지니어링 툴들이 무료 혹은 구매를 통해 사용 가능하다. 가장 발전된 디스어셈블링과 디버깅 툴은 다음과 같다.

- OllyDBG [http://www.ollydbg.de/] (버전 2가 곧 나올 예정)
- IDA Pro Disassembler and Debugger [http://www.hex-rays.com/]
- W32Dasm [http://www.google.com] (오래되었으나 제공하는 기능 중 몇 개는 놀랄만하다)
- SoftICE (2006년 4월부터 진행 중지)
- WinDbg [http://www.microsoft.com/whdc/devtools/debugging/default.mspx]

추가적으로 다른 툴들도 사용된다. 툴 이름과 간단한 설명은 다음과 같다.

- PROTECTiON iD [http://pid.gamecopyworld.com/]
  - 윈도우 실행파일의 packer/encryptor 시그너처와 프로그램 컴파일러를 알려주는데 사용 [http://en.wikipedia.org/wiki/Executable_compression]
- Import REConstructor [http://www.google.com/]
  - 실행파일의 손상된 import table(IAT)의 복구에 사용
- System Internals [http://technet.microsoft.com/en-us/sysinternals/default.aspx]
  - FileMon, RegMon 같은 프로그램으로 대상 프로그램의 동작을 모니터하는데 사용될 수 있다. 다른 대안으로는 모든 프로그램 활동 정보를 제공하는 sandbox가 있다.

리버스 엔지니어링 접근

우리는 리버스 엔지니어가 사용하는 접근을 볼 것이다. 이 장에서 사용되는 디버거는 수정된 버전의 OllyDBG(원래 버전도 잘 동작한다)이다.

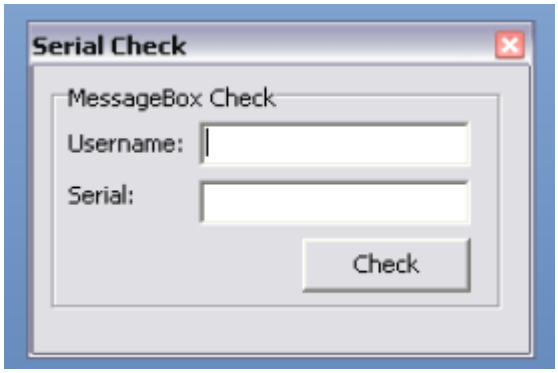
예제 소프트웨어

프로그램 이름: Example.v1.0.exe (시리얼 검사)
Md5sum: 4c78179f07c33e0f9ec8f2b0509bd069
컴파일러: 볼랜드 델파이
다운로드[1]:
http://www.astalavista.com/media/directory/uploads/fcee1505e1287ac6729c610aadd8bedd.zip

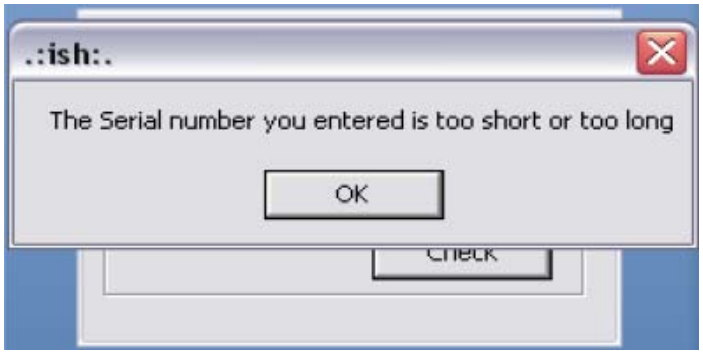
[1] 원래 문서에서는 해당 프로그램을 공개하지 않았다. 역자가 저자에게 메일을 보낸 결과 위의 링크를 가르쳐주었다. 예제 프로그램이 버그가 좀 있어서 공개하지 않았다고 한다. 아무튼 역자의 메일에 친절히 답변해준 저자 George에게 다시 한 번 감사드린다.

프로그램 분석

먼저 우리의 접근 방법을 정하는 것과 어떻게 동작하는지에 대한 더 나은 이해를 위해 프로그램 기능을 분석할 필요가 있다.



보다시피 프로그램의 형태는 간단하다. 주된 기능은 사용자 이름과 시리얼 검사이다. 첫 번째로 텍스트 상자에 임의의 값을 집어넣고, "Check" 버튼을 누르고 프로그램 반응을 살피는 것이다.

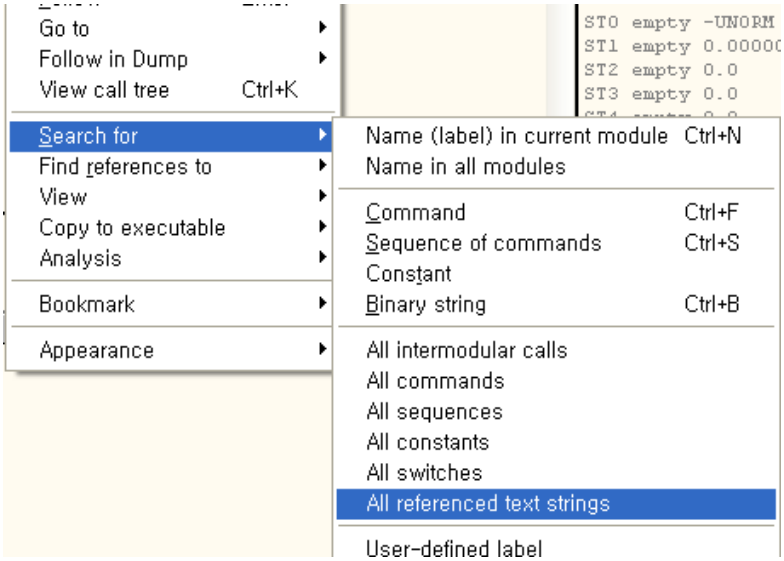


시리얼 검사 알고리즘 전에 주어진 길이 제한 내의 시리얼 문자열이 들어왔는지 검사하는 함수가 있다고 추측할 수 있는 결과를 보여준다.

그런 다음 프로그램이 동작하는 방식에 대한 더 많은 정보를 알기 위해 프로그램을 디스어셈블링하고 디버깅하는 단계로 가자. 앞으로 나올 내용은 리버스 엔지니어가 사용할 방법들과 소프트웨어를 방어하는 몇 가지 제안들이다.

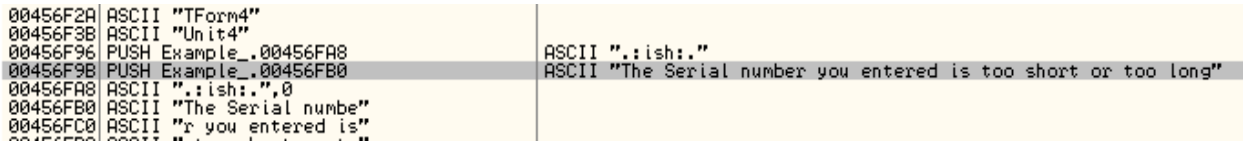
접근 방법 1 (문자열 참조)

첫 번째 단계:
오른쪽 클릭 → 'Search for' 선택 → 'All referenced text strings' 선택



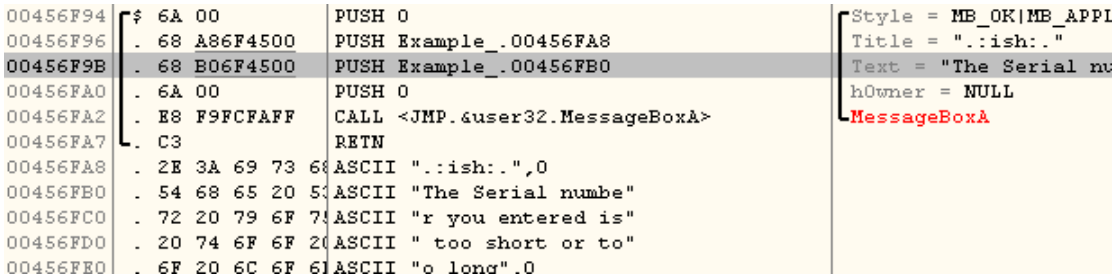
두 번째 단계:

보다시피 메시지 문자열은 다이얼로그 박스와 쉽게 연결되어 있다. 문자열을 더블 클릭해서 다이얼로그 함수로 직접 이동할 수 있다.



세 번째 단계:

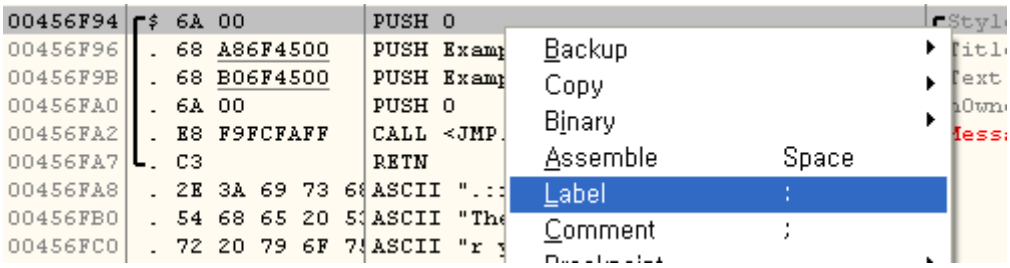
시리얼 검사가 어렵게 코딩되어 있지만, 약간의 경험이 있는 리버스 엔지니어는 이 함수가 어디서 호출되는지 추적할 수 있고 프로그램 흐름을 패치할 수 있다.



네 번째 단계:

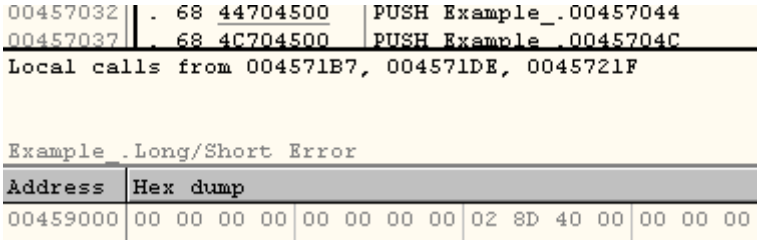
여기서 이 함수의 시작부분에 라벨을 설정(쉽게 참조하기 위해)한다.

오른쪽 클릭 → 'Label' 선택.

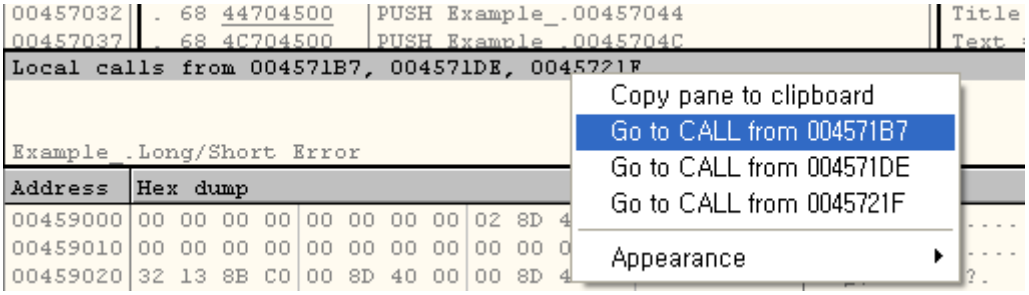


"Long/Short Error"로 라벨 설정.

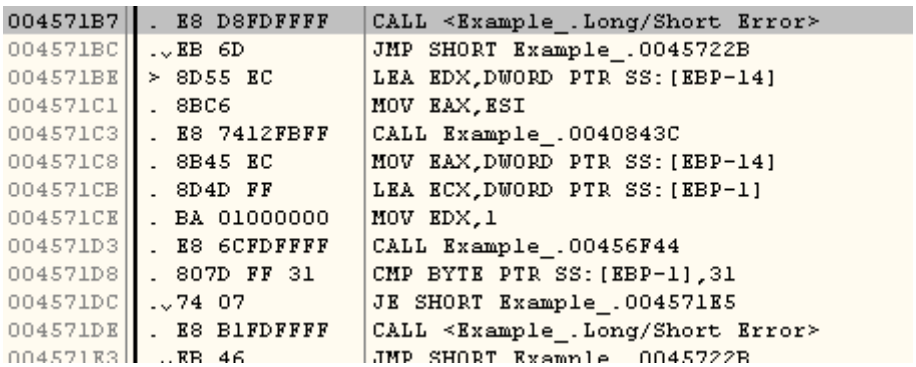
다섯 번째 단계:
밑에서 볼 수 있듯이 이 함수는 서로 가까이 있는 세 개의 다른 주소에서 호출된다.



여섯 번째 단계:
첫 번째 호출한 곳(004571B7)으로 이동한다.
오른쪽 클릭 → 'Go to Call from 004571B7' 선택.



성공적으로 시리얼 검사 알고리즘 내에 도달했다. *만약 호출한 곳에서 시리얼 검사와 관련된 동작을 하지 않는 것으로 보인다면 다른 호출한 주소로 이동해서 살펴보면 된다.*



제안 (접근 방법 1에 대해)

문자열 참조를 통해 민감한 프로그램 함수가 분석 당하지 않기 위해서는 프로그래머는 다음의 과정을 따르면 된다.

- 전역 변수나 배열에 문자열들을 저장해서 필요할 때 참조한다.
  - Pseudo 코드 예제:

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
array[] myMsges = {'The Serial number you entered is too short or too long',
        'The Serial number you entered is not valid',
        'Thank You for registering.'}

//Code omitted

function registrationCheck():
        if(invalid_length) then
                sendMessage(myMsges[0])
        if(invalid_serial) then
                sendMessage(myMsges[1])
        if(valid_serial) then
                sendMessage(myMsges[2])
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

추가적으로 프로그래머는 배열 내부의 문자열을 암호화하고 필요할 때 복호화 할 수 있다 (좋은 암호화 기법을 쓸 필요도 없고 간단한 알고리즘이면 된다).

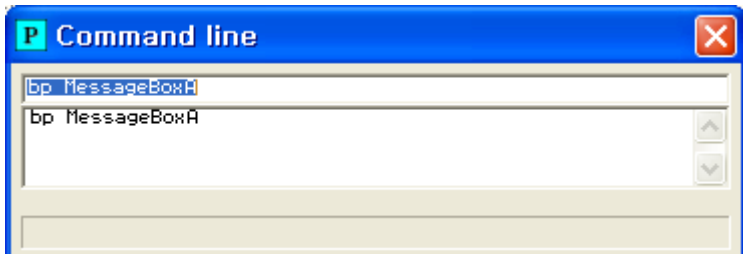
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//This can be done separately.
//Let's assume that the result of this code will be: 'dkg$2 kF2 gkfoaplk'
string thank_you = 'Thank You for registering'

for(each letter in thank_you) do
        add_5_to_ascii_value(letter)
        print thank_you

//program serial check
If(valid_serial) then
        sendMessage(decrypt('dkg$2 kF2 gkfoaplk'))
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

- 파일이나 레지스트리에 문자열을 저장한다.

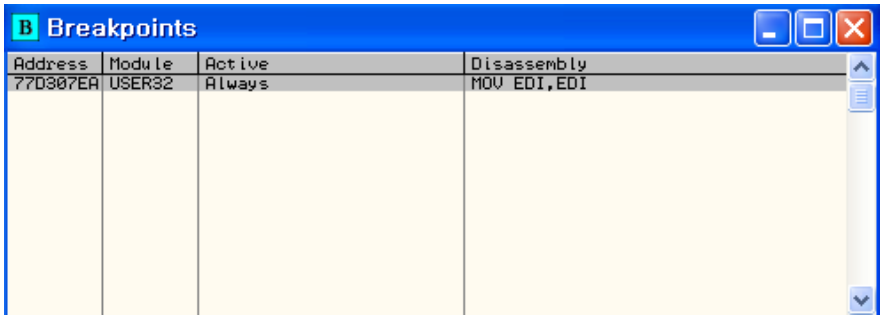
접근 방법 2 (Windows API에 Breakpoint)

이 접근 방법에서는 MessagBoxA API에 breakpoint를 설정한다. 어떤 프로그램은 MessagBoxW, MessageBoxExA, MessageBoxExW를 사용할 수 있다.

첫 번째 단계:
(Ollydbg의 Command Bar plug-in[2] 사용. *Command Bar plug-in은 기본적으로 존재한다.*)
"bp MessageBoxA"를 적고 엔터를 친다.



Breakpoints를 보면 확인 가능하다. 또는 그냥 Ctrl+G 를 누른 후 MessageBoxA를 치면 해당 함수의 시작 부분에 위치하게 된다. 여기서 F2를 눌러 breakpoint를 설정해도 된다.



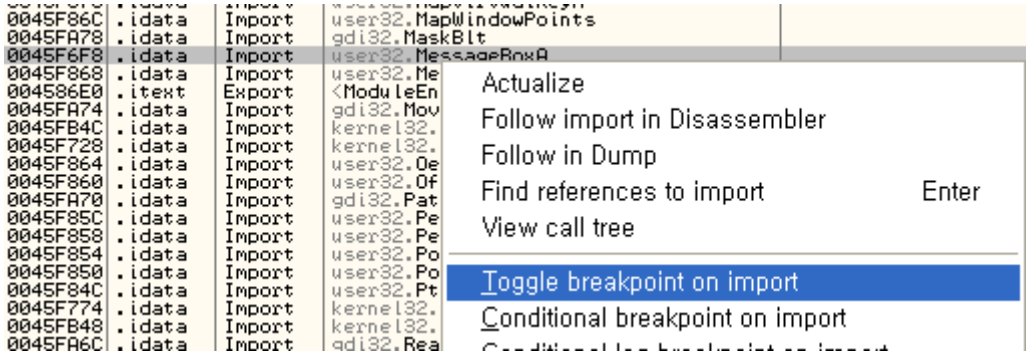
(Ollydbg의 'Names' 창을 사용)
'Alt+E'를 눌러 "Executable Modules" 리스트로 이동 → Executable을 선택하고 'Ctrl+N'을 누름 → 'MessageBoxA' 를 찾음 → 오른쪽 클릭 → "Toggle breakpoint on import" 선택

---

[2] Ollydbg의 plug-in을 이용해서 훨씬 다양하고 편리한 기능을 사용할 수 있다. Plug-in은 사용자가 직접 제작할 수 있는데 자세한 것은 ollydbg 홈페이지를 참고하면 된다. 제작된 plug-in은 다음 사이트에서 다운로드 받을 수 있다.

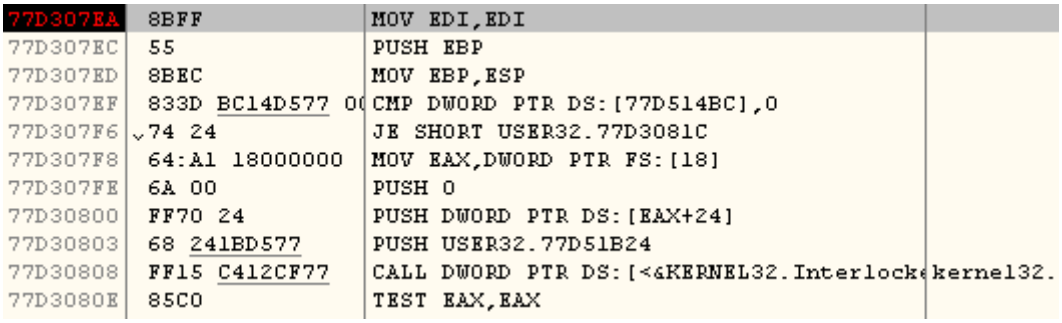
- http://www.openrce.org/downloads/browse/OllyDbg_Plugins

plug-in은 보통 DLL형태인데, ollydbg가 실행되는 같은 폴더에 위치시켜주기만 하면 ollydbg를 실행시켰을 때 메뉴에서 plug-in이 추가된 것을 확인할 수 있다.
저자의 메일에 의하면 ollydbg가 약간의 버그가 있기 때문에 command bar plug-in을 사용했다고 한다.

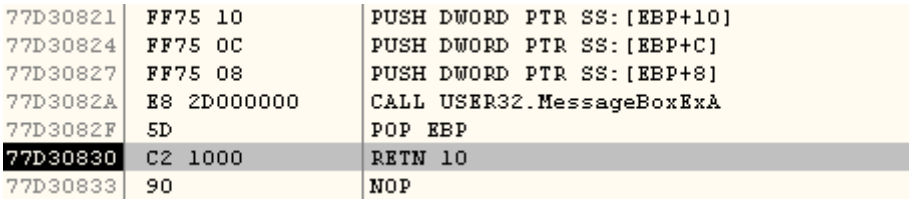


이 경우 앞서 Breakpoints에 설정되었던 것이 사라지게 되어 두 번째 단계에서 breakpoint가 걸리지 않는다. 따라서 역자의 환경에서는 할 필요가 없었다. 이 문제를 저자에게 문의해본 결과 "Toggle breakpoint on import"는 함수 자체가 아닌 함수를 가리키는 포인터의 주소에 break를 설정하는 것이며, 프로그램의 컴파일러에 따라 동작 여부가 달려있다고 한다.

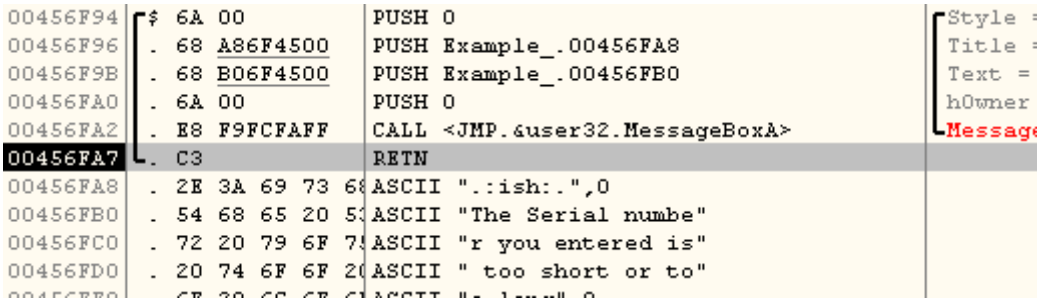
두 번째 단계:
프로그램 실행 → 임의의 데이터 입력 → 'Check' 누름. User32 라이브러리 내의 'MessageBoxA' API에 break가 걸린다.



세 번째 단계:
Return 까지 프로그램 실행(함수 끝날 때 까지 Ctrl+F9 또는 F8 누름)



네 번째 단계:
함수를 빠져 나옴(F8 누름).



보다시피 접근 방법 1에서 세 번째 단계에서와 같은 위치에 도착했다.

제안 (접근 방법 2에 대해)

API에 대한 Breakpoint 설정을 통해 프로그램이 분석되지 않기 위해서는 프로그래머는 API 사용을 제한해야 한다. 프로그램을 최소한의 API 호출로 작성해라. API를 사용하는 대신 자신만의 메시지 박스를 생성해라.

접근 방법 3 (스택 추적)

리버스 엔지니어가 사용할 수 있는 또 다른 흥미 있는 접근 방법은 "스택 추적"이다. 스택 추적이란 스택을 통해 진행 과정을 역 추적하는 기술이다.
CPU에 의해 "CALL <함수>" 명령이 실행되었을 때, Instruction Pointer(EIP)의 값 더하기 다음 명령까지의 바이트 수가 스택에 저장(push)된다(*return address를 의미*). 호출된 함수가 끝나고 "RETN" 명령어가 실행될 때 프로세서는 스택에서 값을 빼와서(pop) 함수를 호출한 다음 곳으로 돌아간다.
예를 들어:

Offset	Opcode
1	PUSH 0
2	CALL 0xF
3	TEST EAX,EAX

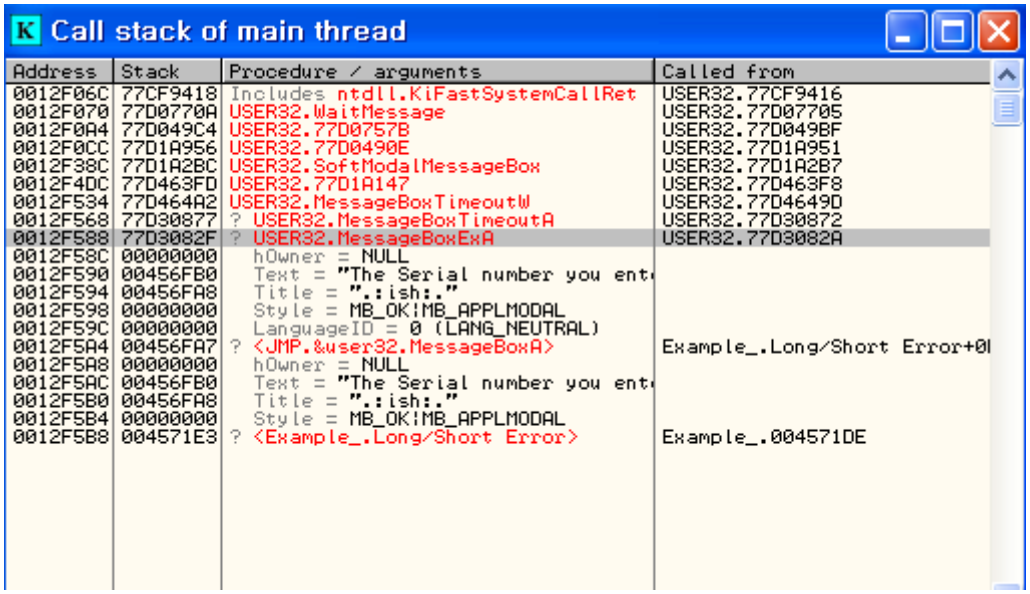
"Call 0xF"가 실행되면, offset 3의 값이 스택에 저장(push)된다.

Offset	Opcode
F	MOV EAX,1
10	RETN
11	NOP

"RETN"이 실행되면, 스택에서 offset 3의 값을 스택에서 가져와서(pop) EIP에 저장한다.

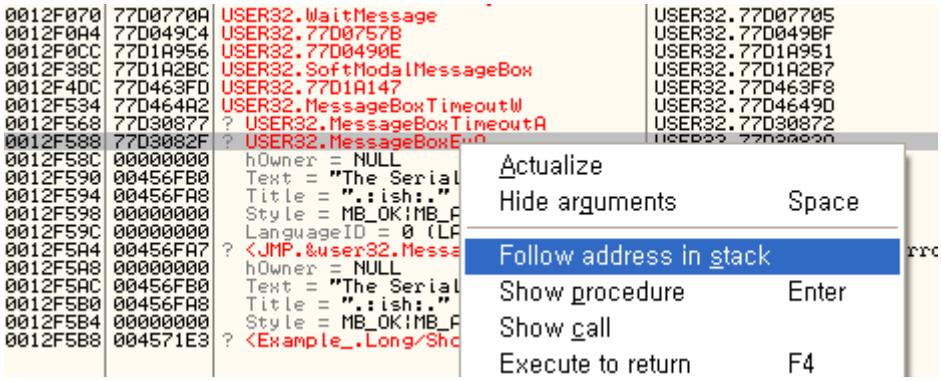
첫 번째 단계:
프로그램 실행 → 임의의 값을 입력 → Check 누름 → 프로그램 중지

두 번째 단계:
"Call stack" 창 열기. *Ollydbg 메뉴의 K를 누르거나 Alt+K를 누르면 된다.*
Call stack 내용을 보면 서로 호출하는 함수들이 있다. 등록 루틴 메인 프로그램을 추적할 수 있는 함수는 MessageBoxExA 이지만 효과적이지는 않다. 그 함수를 어디서 호출하는 지 알아볼 필요가 있다.

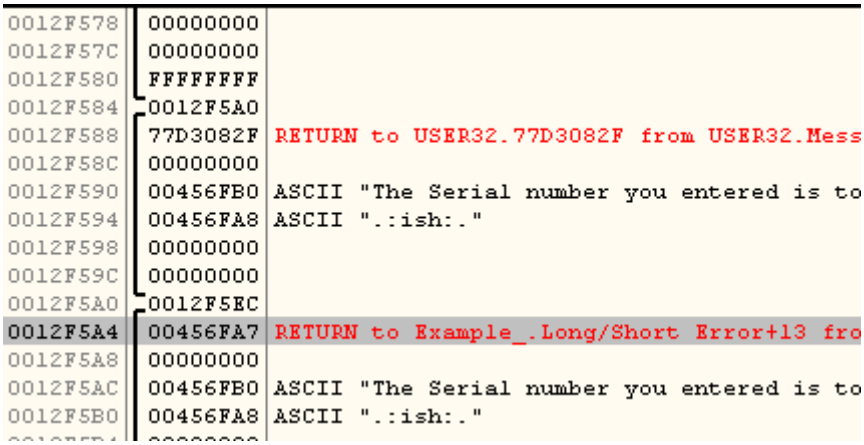


세 번째 단계:

오른쪽 클릭 → 'Follow address in stack' 선택



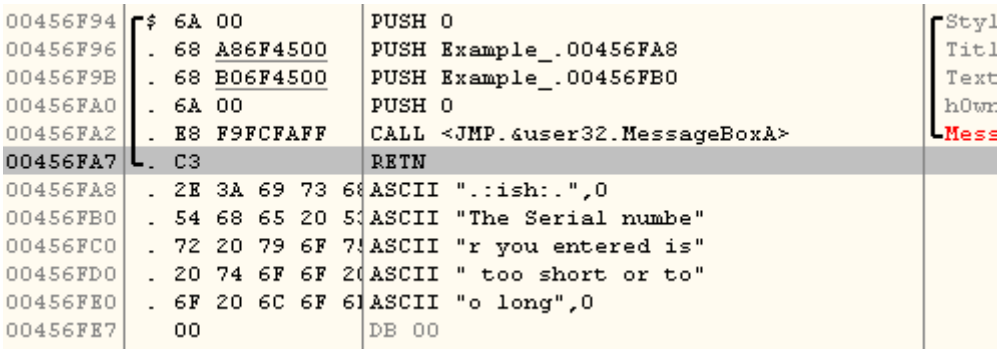
네 번째 단계:



USER32.MessageBoxExA에서 USER32.77D3082F로 리턴한다. USER32.77D3082F는 MessageBoxA 함수이

다(왜 그런지 모르겠다면, user32.dll 내부를 살펴보면 된다).

그러므로 우리가 찾는 함수는 Example_.00456FA7에 존재한다.



제안 (접근 방법 3에 대해)

스택 추적을 막는 것은 어려운 기술이다. 어떤 사람은 프로그램 내의 모든 민감한 함수의 "CALL"과 "RETN" 명령을 "JMP"로 바꿈으로써 할 수 있다고 주장할 수 있다. 이는 "바이너리 코드 obfuscation"이라고 불린다.
코드 obfuscation은 원래 프로그램 바이너리 코드를 변환하는 기술이므로 코드 읽기나 정적 디스어셈블리에 의한 분석을 어렵게 한다. 이 방법이 리버스 엔지니어들을 혼란스럽게 하지만, 소프트웨어를 보호하지는 못한다. 단지 코드 분석을 지연시키기만 한다.
코드 obfuscation의 기본 아이디어는 데이터와 코드 섹션을 합치는 것이다. 또한 obfuscation은 디스어셈블리와 스택 추적을 막기 위해 다음처럼 OPCODE들을 바꾼다.

- 'CALL'을 'PUSH', 'POP', 'RET', 'JMP'로 바꾼다. 그리고 'JMP'를 'PUSH', 'RET'로 바꾼다.
  예를 들어,

원래 코드	Obfuscate된 코드
PUSH 0 CALL 7E450747	PUSH 0 PUSH EIP+<다음 명령어까지의 바이트> JMP 7E450747
MOV EBX, 1 RETN	POP EAX JMP EAX
JMP 00456F94	PUSH 00456F94 RETN

- 'JMP' 분기를 항상 만족하는 조건 분기(예, JE, JNZ, JL)로 바꾼다. 또한 이 방법은 리버스 엔지니어들을 혼란스럽게 하고 다른 코드 섹션으로 보낼 수 있다.

원래 코드	Obfuscate된 코드
JMP 00456F94	MOV EAX, 1

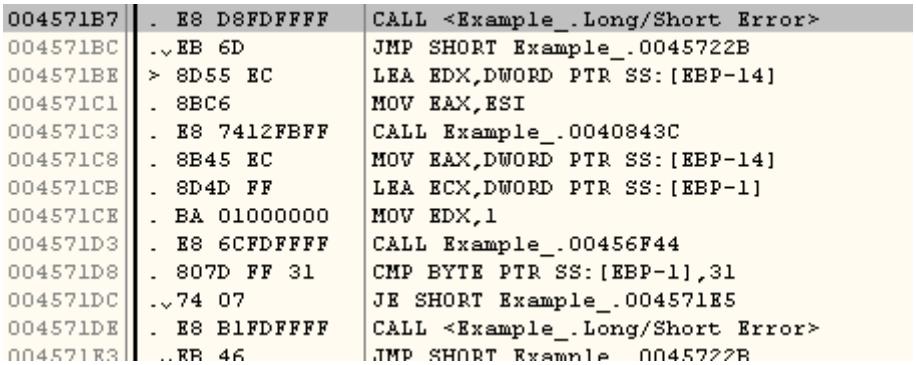
	CMP EAX, 0 JE <JUNK_CODE> JNE 00456F94

- 도달하지 않는 부분에 명령어를 추가한다.
- 오프셋에 대한 직접적인 참조 사용을 피한다(예, JMP 00456F94). 해당 오프셋 값을 혼란스럽게 하기 위해 간단한 계산을 사용하고 나서 호출한다. 예를 들어,

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
MOV EAX, 00456000       ; EAX = 00456000
ADD EAX, 00000F94       ; EAX = 00456F94
JMP EAX                 ; JMP 00456F94
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

바이너리 코드 패치

접근 방법 1의 여섯 번째 단계에서 볼 수 있듯이,



이 코드는 입력된 시리얼 코드가 적합한지 여부를 판단하고 적절하게 등록을 하기 위해 사용자의 실수를 알려주는 실제 알고리즘이다.

리버스 엔지니어가 성공적으로 코드를 패치하고 프로그램 흐름을 조절하기 위해 사용하는 여러 방법이 존재한다. 이를 하기 전에 실제 코드를 분석해야 하고 실제 패치 목표 위치가 어디인지 이해해야 한다.

첫 번째 단계:
화면을 스크롤해서 올려 함수 시작 부분이나 근처에 breakpoint를 설정한다(breakpoint를 설정하려면 해당 명령어를 선택해서 F2를 누르면 된다). 그리고 프로그램을 실행시킨다.

```
00457148  .  55            PUSH EBP
00457149  .  8BEC          MOV EBP,ESP
0045714B  .  33C9          XOR ECX,ECX
0045714D  .  51            PUSH ECX
0045714E  .  51            PUSH ECX
0045714F  .  51            PUSH ECX
00457150  .  51            PUSH ECX
00457151  .  51            PUSH ECX
00457152  .  51            PUSH ECX
00457153  .  51            PUSH ECX
00457154  .  53            PUSH EBX
```

두 번째 단계:
명령어마다 한 스텝씩 따라가면서 이 코드가 어떤 동작을 하는지 이해하도록 노력한다.

아래 그림에서 보듯, 0x0045716F의 CALL 명령어가 텍스트 상자의 "Username:" 안에 입력한 문자열의 포인터를 리턴한다.

코드 섹션:

```
00457160  .  64:FF30       PUSH DWORD PTR FS:[EAX]
00457163  .  64:8920       MOV DWORD PTR FS:[EAX],ESP
00457166  .  8D55 F8       LEA EDX,DWORD PTR SS:[EBP-8]
00457169  .  8B83 6C030000 MOV EAX,DWORD PTR DS:[EBX+36C]
0045716F  .  E8 401AFEFF   CALL Example_.00438BB4
00457174  .  8B55 F8       MOV EDX,DWORD PTR SS:[EBP-8]
00457177  .  B8 A4E54500   MOV EAX,Example_.0045E5A4
0045717C  .  E8 1BD5FAFF   CALL Example_.0040469C
```

현재 명령어:

```
00457169  .  8B83 6C030000 MOV EAX,DWORD PTR DS:[EBX+36C]
0045716F  .  E8 401AFEFF   CALL Example_.00438BB4
00457174  .  8B55 F8       MOV EDX,DWORD PTR SS:[EBP-8]
00457177  .  B8 A4E54500   MOV EAX,Example_.0045E5A4
0045717C  .  E8 1BD5FAFF   CALL Example_.0040469C
00457181  .  8D55 F4       LEA EDX,DWORD PTR SS:[EBP-C]
Stack SS:[0012F5E4]=00E7C6E0, (ASCII "kancho")
EDX=00140608
```

스택:

```
0012F5D4  00000000
0012F5D8  00000000
0012F5DC  00000000
0012F5E0  00000000
0012F5E4  00E7C6E0 ASCII "kancho"
0012F5E8  00000000
0012F5EC  0012F730
0012F5F0  0043A702 RETURN to Example_.0043A702
```

다음 0x0045718F의 CALL 명령어는 "Serial:" 텍스트 상자 안에 사용자가 입력한 문자열의 포인터를 리턴한다.

코드 섹션:

```
00457184  . 8B83 70030000  MOV EAX,DWORD PTR DS:[EBX+370]
0045718A  . E8 251AFEFF    CALL Example_.00438BB4
0045718F  . 8B55 F4        MOV EDX,DWORD PTR SS:[EBP-C]
00457192  . B8 A8E54500    MOV EAX,Example_.0045E5A8
00457197  . E8 00D5FAFF    CALL Example_.0040469C
0045719C  . A1 A8E54500    MOV EAX,DWORD PTR DS:[45E5A8]
004571A1  . 8945 F0        MOV DWORD PTR SS:[EBP-10],EAX
004571A4  . 8B45 F0        MOV EAX,DWORD PTR SS:[EBP-10]
```

현재 명령어:

```
004571B0  > 8BF0           MOV ESI,EAX
Stack SS:[0012F5E0]=00E7C6F8, (ASCII "12345")
EDX=00140608

Address   Hex dump
```

스택:

```
0012F5D4  00000000
0012F5D8  00000000
0012F5DC  00000000
0012F5E0  00E7C6F8  ASCII "12345"
0012F5E4  00E7C6E0  ASCII "kancho"
0012F5E8  00000000
0012F5EC  0012F730
```

다음 코드는 EAX에 사용자가 입력한 시리얼 번호를 로드하고, 입력이 null인지 검사한다. 0x0045E5A8(0x0045719C 오프셋 참조)가 가리키는 값은 오프셋 0x004571A4에서 EAX에 로드된 시리얼 아스키 값이다.

```
If( EAX == null ) {
        //do something
}
```

```
00457192  . B8 A8E54500    MOV EAX,Example_.0045E5A8
00457197  . E8 00D5FAFF    CALL Example_.0040469C
0045719C  . A1 A8E54500    MOV EAX,DWORD PTR DS:[45E5A8]
004571A1  . 8945 F0        MOV DWORD PTR SS:[EBP-10],EAX
004571A4  . 8B45 F0        MOV EAX,DWORD PTR SS:[EBP-10]
004571A7  . 85C0           TEST EAX,EAX
004571A9  .v74 05          JE SHORT Example_.004571B0
004571AB  . 83E8 04        SUB EAX,4
004571AE  . 8B00           MOV EAX,DWORD PTR DS:[EAX]
```

```
0045719C  . A1 A8E54500    MOV EAX,DWORD PTR DS:[45E5A8]
004571A1  . 8945 F0        MOV DWORD PTR SS:[EBP-10],EAX
004571A4  . 8B45 F0        MOV EAX,DWORD PTR SS:[EBP-10]
004571A7  . 85C0           TEST EAX,EAX
004571A9  .v74 05          JE SHORT Example_.004571B0
004571AB  . 83E8 04        SUB EAX,4
004571AE  . 8B00           MOV EAX,DWORD PTR DS:[EAX]
004571B0  > 8BF0           MOV ESI,EAX
Stack SS:[0012F5DC]=00E7C6F8, (ASCII "12345")
EAX=00E7C6F8, (ASCII "12345")
```

레지스터 창:

```
Registers (MMX)
EAX 00E7C6F8 ASCII "12345"
ECX 00000002
EDX 00000000
EBX 00E37180
ESP 0012F5BC
EBP 0012F5EC
ESI 00429804 Example_.00429804
EDI 0012F78C

EIP 004571A4 Example_.004571A4

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
```

아래 보다시피 또 하나의 길이 검사 루틴이 있다. 이번에는 ESI가 사용자가 입력한 시리얼 번호의 길이를 가지고 있으며 이는 1과 비교된다. 길이가 1이면 "Long/Short Error"가 호출(접근 방법 1의 네 번째 단계 참조)된다.

코드 섹션:

```
004571AB   . 83E8 04        SUB EAX,4
004571AE   . 8B00           MOV EAX,DWORD PTR DS:[EAX]
004571B0   > 8BF0           MOV ESI,EAX
004571B2   . 83FE 01        CMP ESI,1
004571B5   .v75 07          JNZ SHORT Example_.004571BE
004571B7   . E8 D8FDFFFF    CALL Example_.00456F94
004571BC   .vEB 6D          JMP SHORT Example_.0045722B
004571BE   > 8D55 EC        LEA EDX,DWORD PTR SS:[EBP-14]
```

레지스터 창:

```
Registers (MMX)
EAX 00000005
ECX 00000002
EDX 00000000
EBX 00E37180
ESP 0012F5BC
EBP 0012F5EC
ESI 00000005
EDI 0012F78C

EIP 004571B5 Example_.004571B5

C 0  ES 0023 32bit 0(FFFFFFFF)
```

아래 하이라이트된 코드는 십진수로의 시리얼 넘버 길이의 첫 번째 문자열 값(*예를 들어 길이가 28이면 '2'*)과 '1'(아스키 값 0x31)을 비교한다. 어떻게 길이가 아스키 문자열로 바뀌는지 궁금한 사람들을 위해 0x4571C3에서의 CALL 명령을 따라가면 다음 루프를 볼 수 있다.

```
0040840D   |> 31D2              /XOR EDX,EDX
```

```
0040840F   |. F7F1            |DIV ECX
00408411   |. 4E              |DEC ESI
00408412   |. 80C2 30         |ADD DL,30
00408415   |. 80FA 3A         |CMP DL,3A
00408418   |. 72 03           |JB SHORT Example_.0040841D
0040841A   |. 80C2 07         |ADD DL,7
0040841D   |> 8816             |MOV BYTE PTR DS:[ESI],DL
0040841F   |. 09C0            |OR EAX,EAX
00408421   |.^75 EA            ₩JNZ SHORT Example_.0040840D
```

```
0040840D | > 31D2          | XOR EDX,EDX
0040840F | . F7F1          | DIV ECX
00408411 | . 4E            | DEC ESI
00408412 | . 80C2 30       | ADD DL,30
00408415 | . 80FA 3A       | CMP DL,3A
00408418 | .v72 03         | JB SHORT Example_.0040841D
0040841A | . 80C2 07       | ADD DL,7
0040841D | > 8816          | MOV BYTE PTR DS:[ESI],DL
0040841F | . 09C0          | OR EAX,EAX
00408421 | .^75 EA         | JNZ SHORT Example_.0040840D
```

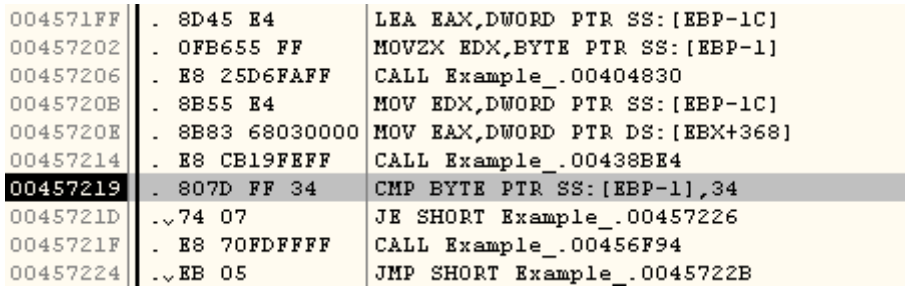
코드 섹션:

```
004571BE | > 8D55 EC       | LEA EDX,DWORD PTR SS:[EBP-14]
004571C1 | . 8BC6          | MOV EAX,ESI
004571C3 | . E8 7412FBFF   | CALL Example_.0040843C
004571C8 | . 8B45 EC       | MOV EAX,DWORD PTR SS:[EBP-14]
004571CB | . 8D4D FF       | LEA ECX,DWORD PTR SS:[EBP-1]
004571CE | . BA 01000000   | MOV EDX,1
004571D3 | . E8 6CFDFFFF   | CALL Example_.00456F44
004571D8 | . 807D FF 31    | CMP BYTE PTR SS:[EBP-1],31
004571DC | .v74 07         | JE SHORT Example_.004571E5
004571DE | . E8 B1FDFFFF   | CALL Example_.00456F94
```

현재 명령어:

```
004571D8 | . 807D FF 31    | CMP BYTE PTR SS:[EBP-1],31
004571DC | .v74 07         | JE SHORT Example_.004571E5
004571DE | . E8 B1FDFFFF   | CALL Example_.00456F94
004571E3 | .vEB 46         | JMP SHORT Example_.0045722B
004571E5 | > 8D55 E8       | LEA EDX,DWORD PTR SS:[EBP-18]
004571E8 | . 8BC6          | MOV EAX,ESI
Stack SS:[0012F5EB]=35 ('5')
```

다음은 위의 것과 비슷하지만 두 번째 숫자를 검사한다. 이런 경우에 두 번째 숫자는 '4', hex 값으로 0x34와 같아야 한다.



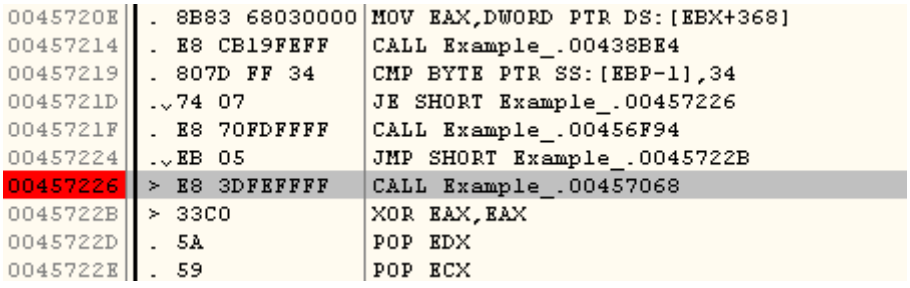
그래서 지금까지의 프로그램 코드는 다음과 같이 생겼다.

```
char first = getChar(length,1,?); //Get first character
if (first != '1') {
        char second = getChar(length,2,?); //Get second character

        if(second != '4') {
                //continue with serial check
        }
        else {
                sendLongShortError();
        }
}
else {
        sendLongShortError();
}
```

참고: '?' 문자는 리턴된 값 데이터 타입의 알려지지 않은 값을 의미한다.

대부분은 입력한 시리얼 번호의 길이가 14이지 않기 때문에 F9를 눌러서 다시 시리얼 번호를 입력한다 (*이번에는 시리얼 번호를 '123456789abcde'를 입력했다*).

세 번째 단계:
0x00457226의 CALL을 따라가보면(F7을 눌러) 아래 그림과 같은 코드를 볼 수 있다.



네 번째 단계:
아래 코드를 보자.

```
00457068   /$ 53                PUSH EBX
00457069   |. 56                PUSH ESI
0045706A   |. 57                PUSH EDI
0045706B   |. 55                PUSH EBP
0045706C   |. 83C4 F8           ADD ESP,-8
0045706F   |. BB 01000000       MOV EBX,1
00457074   |. BE ACE54500       MOV ESI,Example_.0045E5AC                ;  ASCII "123456789abcde"
00457079   |> 8BCE              /MOV ECX,ESI
0045707B   |. 8BD3              |MOV EDX,EBX
0045707D   |. A1 A8E54500       |MOV EAX,DWORD PTR DS:[45E5A8]
00457082   |. E8 BDFEFFFF       |CALL Example_.00456F44
00457087   |. 43                |INC EBX
00457088   |. 46                |INC ESI
00457089   |. 83FB 0F           |CMP EBX,0F
0045708C   |.^75 EB             ₩JNZ SHORT Example_.00457079
0045708E   |. A1 A4E54500       MOV EAX,DWORD PTR DS:[45E5A4]
00457093   |. 894424 04         MOV DWORD PTR SS:[ESP+4],EAX
00457097   |. 8B4424 04         MOV EAX,DWORD PTR SS:[ESP+4]
0045709B   |. 85C0              TEST EAX,EAX
0045709D   |. 74 05             JE SHORT Example_.004570A4
0045709F   |. 83E8 04           SUB EAX,4
004570A2   |. 8B00              MOV EAX,DWORD PTR DS:[EAX]
004570A4   |> 85C0              TEST EAX,EAX
004570A6   |. 7E 16             JLE SHORT Example_.004570BE
004570A8   |. BB 01000000       MOV EBX,1
004570AD   |> 8B15 A4E54500     /MOV EDX,DWORD PTR DS:[45E5A4]
004570B3   |. 0FB6541A FF       |MOVZX EDX,BYTE PTR DS:[EDX+EBX-1]
004570B8   |. 03EA              |ADD EBP,EDX
004570BA   |. 43                |INC EBX
004570BB   |. 48                |DEC EAX
004570BC   |.^75 EF             ₩JNZ SHORT Example_.004570AD
004570BE   |> BB 0E000000       MOV EBX,0E
004570C3   |. B8 ACE54500       MOV EAX,Example_.0045E5AC                ;  ASCII "123456789abcde"
004570C8   |. BA BCE54500       MOV EDX,Example_.0045E5BC
004570CD   |> 0FB608            /MOVZX ECX,BYTE PTR DS:[EAX]
004570D0   |. 890A              |MOV DWORD PTR DS:[EDX],ECX
```

```
004570D2   |. 83C2 04            |ADD EDX,4
004570D5   |. 40                 |INC EAX
004570D6   |. 4B                 |DEC EBX
004570D7   |.^75 F4              ₩JNZ SHORT Example_.004570CD
004570D9   |> 803D ACE54500 >/CMP BYTE PTR DS:[45E5AC],7B
004570E0   |. 74 07              |JE SHORT Example_.004570E9
004570E2   |. BF 01000000        |MOV EDI,1
004570E7   |. EB 46              |JMP SHORT Example_.0045712F
004570E9   |> 8BC5               |MOV EAX,EBP
004570EB   |. B9 0A000000        |MOV ECX,0A
004570F0   |. 99                 |CDQ
004570F1   |. F7F9               |IDIV ECX
004570F3   |. 0FB605 ADE5450>|MOVZX EAX,BYTE PTR DS:[45E5AD]
004570FA   |. 3BD0               |CMP EDX,EAX
004570FC   |. 75 06              |JNZ SHORT Example_.00457104
004570FE   |. 830424 02          |ADD DWORD PTR SS:[ESP],2
00457102   |. EB 2B              |JMP SHORT Example_.0045712F
00457104   |> BB 0C000000        |MOV EBX,0C
00457109   |. BE ADE54500        |MOV ESI,Example_.0045E5AD                ;  ASCII "23456789abcde"
0045710E   |> 0FB606             |/MOVZX EAX,BYTE PTR DS:[ESI]
00457111   |. B9 0A000000        ||MOV ECX,0A
00457116   |. 33D2               ||XOR EDX,EDX
00457118   |. F7F1               ||DIV ECX
0045711A   |. 8BCA               ||MOV ECX,EDX
0045711C   |. 83F9 0E            ||CMP ECX,0E
0045711F   |. 73 0A              ||JNB SHORT Example_.0045712B
00457121   |. 83F9 01            ||CMP ECX,1
00457124   |. 76 05              ||JBE SHORT Example_.0045712B
00457126   |. E8 05FFFFFF        ||CALL Example_.00457030
0045712B   |> 46                 ||INC ESI
0045712C   |. 4B                 ||DEC EBX
0045712D   |.^75 DF              |₩JNZ SHORT Example_.0045710E
0045712F   |> 83FF 01            |CMP EDI,1
00457132   |.^75 A5              ₩JNZ SHORT Example_.004570D9
00457134   |. 8B0424             MOV EAX,DWORD PTR SS:[ESP]
00457137   |. 83E8 02            SUB EAX,2
0045713A   |. 75 05              JNZ SHORT Example_.00457141
0045713C   |. E8 A7FEFFFF        CALL Example_.00456FE8
00457141   |> 59                 POP ECX
```

```
00457142   |. 5A              POP EDX
00457143   |. 5D              POP EBP
00457144   |. 5F              POP EDI
00457145   |. 5E              POP ESI
00457146   |. 5B              POP EBX
00457147   ₩. C3              RETN
```

이 코드는 실제 시리얼 번호 검사를 수행한다. 디버깅 시 코드를 분석할 때 볼 수 있듯이, 0x0045713C에 위치한 CALL 명령을 실행시키지 않도록 하는 많은 jump 명령을 볼 수 있다. 보통 원하는 결과가 나올 수 있는 여러 방법들이 존재한다. 이러한 방법 중에는 패치, 분석, 재구성 또는 코드(어셈블리) 복사(ripping) 들을 포함한다. 이 프로그램에서는 단지 몇 개의 방법들이 있다. 보다시피 위의 코드는 시리얼 키를 검사하고 사용자와 시리얼 키 간의 일치 여부에 대해 알려준다.
다음 방법은 실제로는 적용될 수 없을지 모르지만 어떻게 리버스 엔지니어들이 일해야 하는지에 대해 기본적이고 간단한 아이디어를 제공한다.

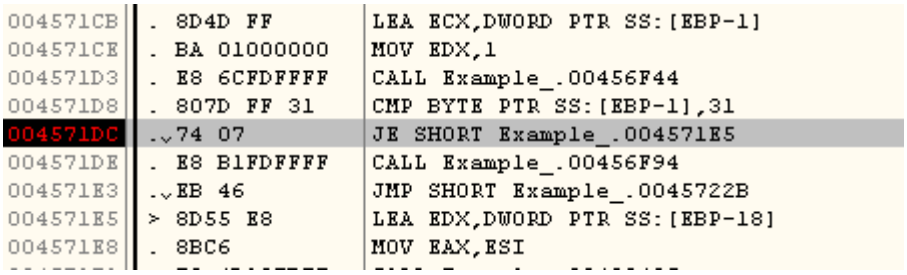
접근 방법 1 (분기 패치)

프로그램 흐름을 패치하는 하나의 방법은 조건 분기 수정을 통해서이다.

사용자가 입력한 시리얼 키가 올바르지 않다는 것을 결정하는 시리얼 키 검사 알고리즘이 존재하는곳은 많다. 여기는,

첫 번째 검사:
위의 바이너리 분석에서 볼 수 있듯이 함수는 시리얼 키 길이를 아스키 문자열로 바꾸고 첫 번째 문자가 아스키로 '1'이며 hex로는 '0x31'인지 검사한다.

간단한 패치는,
0x004571DC의 opcode를 더블 클릭하고
"JE SHORT 004571E5"를 "JMP SHORT 004571E5"로 바꾼다.
그러면 0x004571DE에 위치한 CALL은 절대 호출되지 않는다.



```
004571CB | . 8D4D FF       | LEA ECX,DWORD PTR SS:[EBP-1]
004571CE | . BA 01000000   | MOV EDX,1
004571D3 | . E8 6CFDFFFF   | CALL Example_.00456F44
004571D8 | . 807D FF 31    | CMP BYTE PTR SS:[EBP-1],31
004571DC |   EB 07         | JMP SHORT Example_.004571E5
004571DE | . E8 B1FDFFFF   | CALL Example_.00456F94
004571E3 | . EB 46         | JMP SHORT Example_.0045722B
004571E5 | > 8D55 E8       | LEA EDX,DWORD PTR SS:[EBP-18]
004571E8 | . 8BC6          | MOV EAX,ESI
004571EA | . E8 4D12FBFF   | CALL Example_.0040843C
```

두 번째 검사:

0x0045721D의 조건 분기에서도 첫 번째 검사와 동일하게 적용한다.

```
0045720B | . 8B55 E4       | MOV EDX,DWORD PTR SS:[EBP-1C]
0045720E | . 8B83 68030000 | MOV EAX,DWORD PTR DS:[EBX+368]
00457214 | . E8 CB19FEFF   | CALL Example_.00438BE4
00457219 | . 807D FF 34    | CMP BYTE PTR SS:[EBP-1],34
0045721D | . 74 07         | JE SHORT Example_.00457226
0045721F | . E8 70FDFFFF   | CALL Example_.00456F94
00457224 | . EB 05         | JMP SHORT Example_.0045722B
00457226 | > E8 3DFEFFFF   | CALL Example_.00457068
0045722B | > 33C0          | XOR EAX,EAX
0045722D | . 5A            | POP EDX
```

```
0045720B | . 8B55 E4       | MOV EDX,DWORD PTR SS:[EBP-1C]
0045720E | . 8B83 68030000 | MOV EAX,DWORD PTR DS:[EBX+368]
00457214 | . E8 CB19FEFF   | CALL Example_.00438BE4
00457219 | . 807D FF 34    | CMP BYTE PTR SS:[EBP-1],34
0045721D |   EB 07         | JMP SHORT Example_.00457226
0045721F | . E8 70FDFFFF   | CALL Example_.00456F94
00457224 | . EB 05         | JMP SHORT Example_.0045722B
00457226 | > E8 3DFEFFFF   | CALL Example_.00457068
0045722B | > 33C0          | XOR EAX,EAX
0045722D | . 5A            | POP EDX
```

그러므로 두 번째 단계에서의 코드를 바이너리 코드 패치는 다음과 같이 바꾼다.

```
char first = getChar(length,1,?); //Get first character
if(true) { //This is always true
        char second = getChar(length,2,?); //Get second character
        if(true) { //This is always true
                //continue with serial check
        }
        else {
                sendLongShortError(); //This is never called
        }
}
else {
        sendLongShortError(); //This is never called
}
```

세 번째 검사:
마찬가지로 조건 분기를 무조건 분기(JMP)로 패치한다.

```
004570D5  . 40            INC EAX
004570D6  . 4B            DEC EBX
004570D7  .^75 F4         JNZ SHORT Example_.004570CD
004570D9  > 803D ACE54500 CMP BYTE PTR DS:[45E5AC],7B
004570E0  .v74 07         JE SHORT Example_.004570E9
004570E2  . BF 01000000   MOV EDI,1
004570E7  .vEB 46         JMP SHORT Example_.0045712F
004570E9  > 8BC5          MOV EAX,EBP
004570EB  . B9 0A000000   MOV ECX,0A
004570F0  . 99            CDQ
```

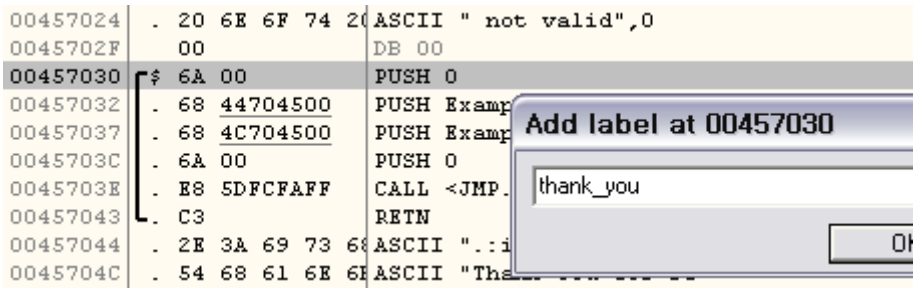
```
004570D5  . 40            INC EAX
004570D6  . 4B            DEC EBX
004570D7  .^75 F4         JNZ SHORT Example_.004570CD
004570D9  > 803D ACE54500 CMP BYTE PTR DS:[45E5AC],7B
004570E0    vEB 07        JMP SHORT Example_.004570E9
004570E2  . BF 01000000   MOV EDI,1
004570E7  .vEB 46         JMP SHORT Example_.0045712F
004570E9  > 8BC5          MOV EAX,EBP
004570EB  . B9 0A000000   MOV ECX,0A
004570F0  . 99            CDQ
```

일반적으로 이런 방식으로 하면 된다. 약간의 버그가 있지만 기본적인 아이디어는 이해하리라 믿는다.

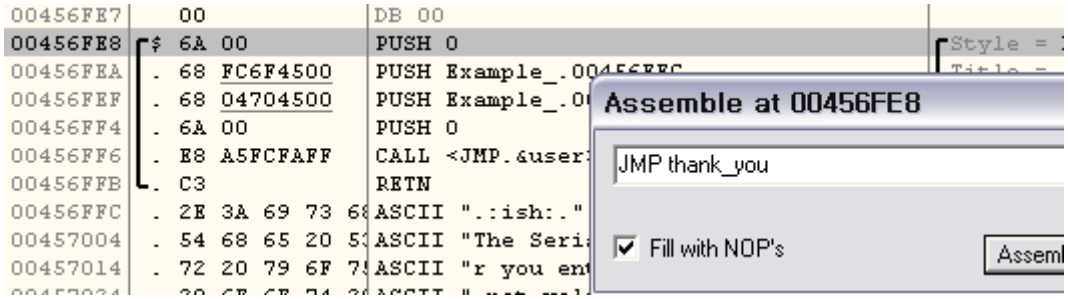
접근 방법 2 (함수 대체)

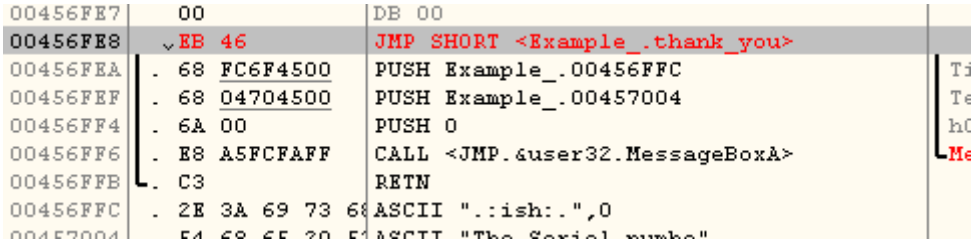
더 간단한 방법은 아래와 같이 에러 메시지 함수를 수정하여 이를 성공할 때 호출되는 함수를 가리키도록 하는 것이다.

먼저 코드 부분을 검색해서 "thank you for registering" 문자열을 출력하는 부분을 찾아 라벨을 'thank_you'로 등록한다.



그리고 나서 에러 메시지를 출력하는 부분을 찾아 시작부분에 스페이스 바를 눌러 코드를 'JMP thank_you'로 바꾼다.





참고: 이는 프로그래머가 모든 것을 한 함수 내에 다 넣지 않을 만큼 현명하다면 제대로 동작하지 않을 것이다.

시리얼 생성 (키 생성)

여기서는 "크래커"가 프로그램 코드를 분석하고, 입력한 시리얼이 옳은 지를 검사하는 것이 아니라 항상 올바른 시리얼 키(외부 제약의 조건 없이)를 생성하는 방법으로 등록 알고리즘을 재구성한다. 시리얼 생성 알고리즘 재구성에 대해 아래 몇 가지 방법이 사용된다.

코드 재구성

리버스 엔지니어가 저수준 어셈블리어를 고수준 프로그래밍 언어(C, C++, .NET, JAVA 등)로 변환하는방법으로 함수나 함수 집합의 동작을 이해하기 위한 알고리즘의 분석(보통 디버깅을 통해).

예제:

- 저수준:

```
004570AD |> /MOV EDX,DWORD PTR DS:[45E5A4] ; Load username string in EDX
004570B3 | |MOVZX EDX,BYTE PTR DS:[EDX+EBX-1] ; Get letter in position EBX-1 (in each
                                                ; loop the pointer is incr by 1)
004570B8 |. |ADD EBP,EDX ; Add the hexadecimal ASCII value of the letter in EBP (UserCount)
004570BA |. |INC EBX ; Increase the pointer (EBX)
004570BB |. |DEC EAX ; Decrease the loop counter
004570BC |.^ ₩JNZ SHORT Example_.004570AD ; Stop branching only when the loop
                                          ; counter reaches zero(0)
004570BE |> MOV EBX,0E
```

```
004570C3 |. MOV EAX,Example_.0045E5AC
004570C8 |. MOV EDX,Example_.0045E5BC
004570CD |> /MOVZX ECX,BYTE PTR DS:[EAX] ; Get the ASCII char stored in memory
                                          ; at EAX (Serial string pointer)
004570D0 |. |MOV DWORD PTR DS:[EDX],ECX ; Store it an array of integer
                                          ; (see next operation)?
004570D2 |. |ADD EDX,4 ; Move 4 bytes to the right => An array of 32bit Integer values
004570D5 |. |INC EAX ; Move memory pointer one(1) byte to the right
004570D6 |. |DEC EBX ; Decrease loop counter
004570D7 |.^ ₩JNZ SHORT Example_.004570CD ; Stop branching when loop
                                          ; counter reaches zero(0)
004570D9 |> CMP BYTE PTR DS:[45E5AC],7B ; Compare first character
                                          ; from the ASCII value with 0x7B ( "{" )
//Code omitted
```

- 고수준 (자바)

```
String username = getUsername();
int sum = 0;
for(int i = 0;i < username.length(); i++) {
        sum += username.charAt(i);
}
String serial = getSerial();
int[] array = new int[255];
if(serial.length()<=255) { //Well, Java is safe but we don't need exceptions popping around.
        for(int i = 0;i < serial.length(); i++) {
                array[i] = serial.charAt(i);
        }
}
If(serial.charAt(0) == '{') {
//Code omitted
```

코드 복사 (ripping)

이는 프로그램의 바이너리 코드를 다른 프로그램이나 직접 어셈블리 코딩을 지원하는 상위 프로그래밍 언어 안에 복사하는 여러 방법들을 말한다. 코드 복사 방법은 디버깅에 드는 시간과 노력이 코드 재구성에 비해 상당히 줄어들기 때문에 코드 재구성과는 상관 없다.

예제:

- 저수준:

```
004570AD |> /MOV EDX,DWORD PTR DS:[45E5A4] ; Load username string in EDX
004570B3 | |MOVZX EDX,BYTE PTR DS:[EDX+EBX-1] ; Get letter in position EBX-1 (in each
                                                ; loop the pointer is incr by 1)
004570B8 |. |ADD EBP,EDX ; Add the hexadecimal ASCII value of the letter in EBP (UserCount)
004570BA |. |INC EBX ; Increase the pointer (EBX)
004570BB |. |DEC EAX ; Decrease the loop counter
004570BC |.^ ₩JNZ SHORT Example_.004570AD ; Stop branching only
                                          ; when the loop counter reaches zero(0)
004570BE |> MOV EBX,0E
004570C3 |. MOV EAX,Example_.0045E5AC
004570C8 |. MOV EDX,Example_.0045E5BC
004570CD |> /MOVZX ECX,BYTE PTR DS:[EAX] ; Get the ASCII char stored
                                          ; in memory at EAX (Serial string pointer)
004570D0 |. |MOV DWORD PTR DS:[EDX],ECX ; Store it an array of integer
                                          ; (see next operation)?
004570D2 |. |ADD EDX,4 ; Move 4 bytes to the right => An array of 32bit Integer values
004570D5 |. |INC EAX ; Move memory pointer one(1) byte to the right
004570D6 |. |DEC EBX ; Decrease loop counter
004570D7 |.^ ₩JNZ SHORT Example_.004570CD ; Stop branching when
                                          ; loop counter reaches zero(0)
004570D9 |> CMP BYTE PTR DS:[45E5AC],7B ; Compare first character
                                          ; from the ASCII value with 0x7B ( "{" )
//Code omitted
```

- 고 수준 복사(High Level Rip):

```
//Code omitted
getUsername(username);
getPassword(password);
user_length := length(username);
pass_length := length(password);
asm
        @loop1:
        MOV EAX,user_length
        MOV EBX,1
        MOV EDX,&username
        MOVZX EDX,BYTE [EDX+EBX-1]
        ADD EBP,EDX
        INC EBX
        DEC EAX
```

```
        JNZ @loop1
        //Code omitted
end;
```

기타

라이센스 서비스의 사용은 리버스 엔지니어와 키 생성의 위험을 증가시킨다. 나는 당신의 소프트웨어에 다른 회사의 프로그램 컴포넌트와 함께 구현하는 것을 반대하지 않으며 그것이 보안 상의 결점이라고 생각하지 않는다. 이를 보안 위험요소라고 하는 것은 구현 상의 불완전성과 소프트웨어를 이해하기 위해 보낸 시간이 부족해서이다.