



BEISLAB FOR SECURITY SINCE 2001

[Immunity Debugger & Python (Part 1)]

Written by Osiris (email, msn – mins4416@naver.com)

by beistlab(<http://beist.org>)

Synopsis

Immunity Debugger는 Python을 플러그인 형태로 지원하고 있습니다. 이 2개를 연동할 경우 강력한 Reverse Engineering 환경을 구축할 수 있습니다. 본 문서에서는 Immunity Debugger + Python 구조에 대해서 다루고 있습니다. 먼저 Immunity Debugger에 대해 간단하게 설명하고 예제 프로그램의 문제를 해결하기 위해 만들어진 Python script를 분석하도록 하겠습니다. 우리는 script분석을 통해서 script작성법과 module에 들어 있는 여러 종류의 method에 대한 사용법을 배울 수 있을 것입니다. 예제프로그램의 문제 해결을 위해 만들어진 script는 Immunity Debugger Forum의 f3님이 만드신 것을 인용하였습니다. 본 문서를 큰 어려움 없이 읽기 위해서는 어셈블리와 Python에 대해서 기초 지식은 알고 있어야 합니다.

Contents

0x01. Immunity Debugger

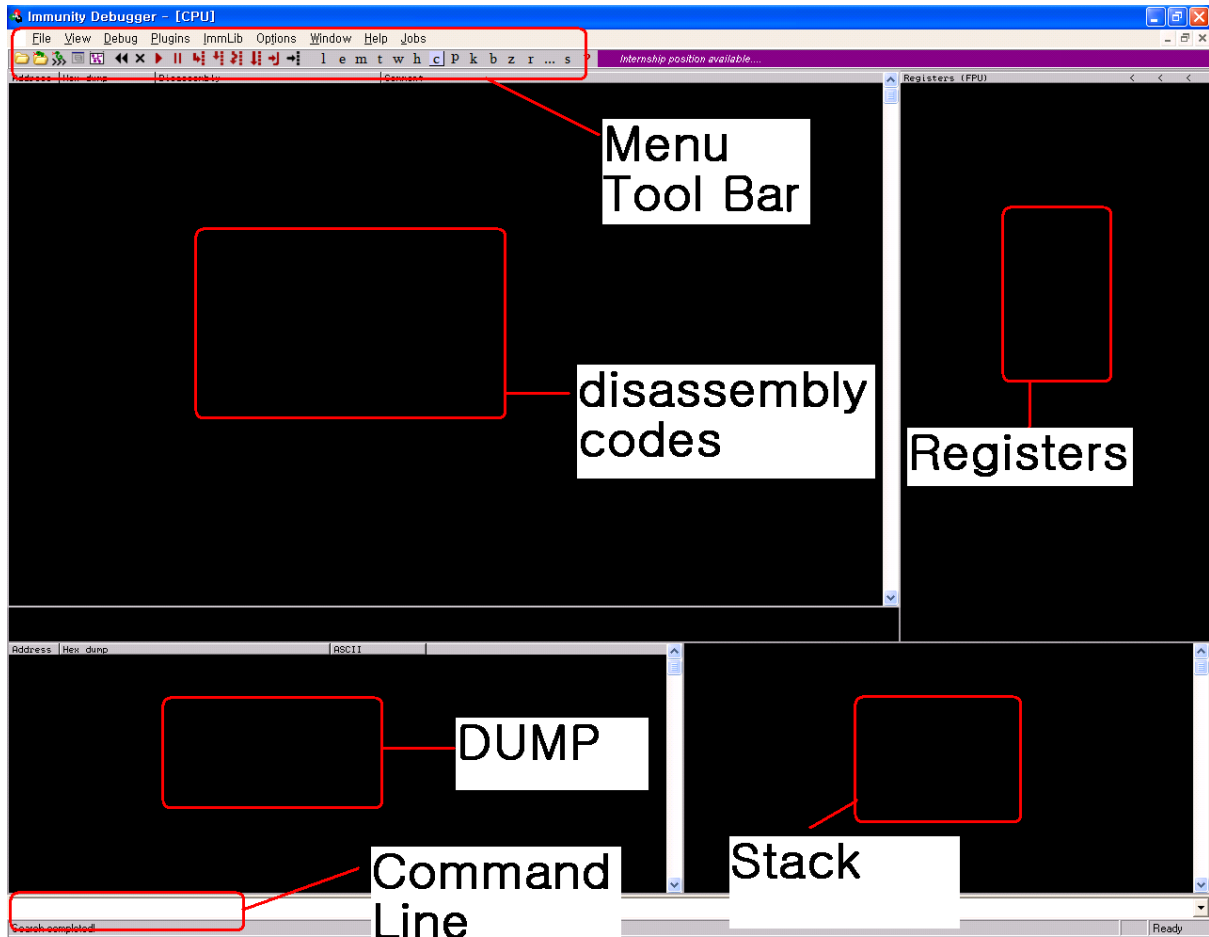
0x02. Python Script 1

0x03. Python Script 2

0x04. 참고사이트 & 참고문헌

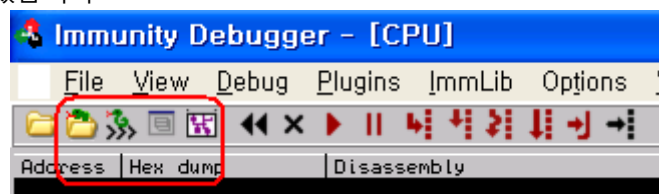
0x01. Immunity Debugger

Immunity Debugger는 Tool Bar나 작업영역 등 Olly Debugger와 매우 비슷한 모양입니다. Immunity Debugger는 Olly Debugger처럼 GUI기반이며 command line을 가지고 있습니다.



[그림1-1. Immunity Debugger를 실행한 모습]

[그림1-1]에서 보시는 것처럼 Olly Debugger와 Immunity Debugger는 상당히 흡사한 것을 알 수 있습니다. 이렇게 비슷한 모양을 하고 있지만 Immunity Debugger에는 Olly Debugger엔 없는 막강한 기능을 갖고 있습니다.

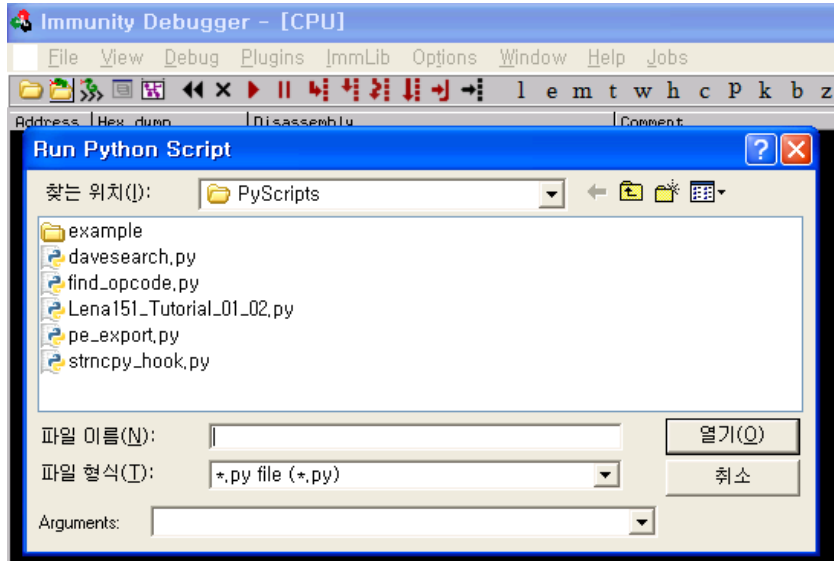


[그림1-2. Olly Debugger에는 없는 Immunity Debugger만의 Tool Bar Icon들]



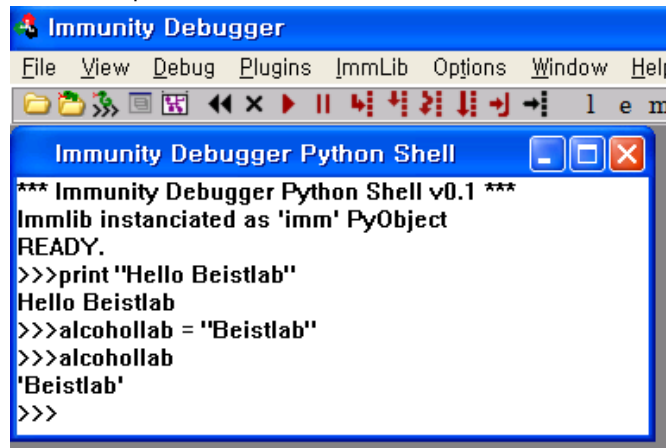
[그림1-3. Olly Debugger Tool Bar Icon]

[그림1-2]와 [그림1-3]을 비교해보면 [그림1-3]에는 없지만 [그림1-2]에는 존재하는 Tool Bar Icon을 볼 수 있습니다. [그림1-2]를 보면 빨간색 테두리에 4개의 icon이 있습니다. 각각의 특징을 알아 보도록 하겠습니다.



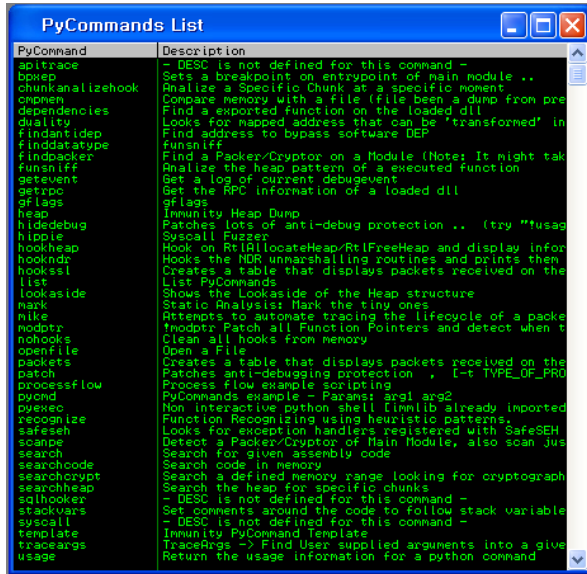
[그림1-4. Run Python Script]

첫 번째 icon은 python script실행 icon입니다. [그림1-4]에서 보이는 것처럼 python으로 만들어진 script를 실행시켜줍니다. Script에 대해서는 있다가 뒤에서 알아보도록 하겠습니다.



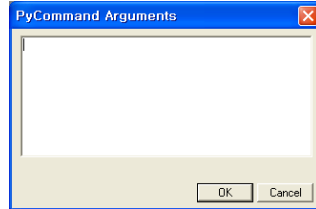
[그림1-5. Immunity Debugger Python Shell]

두 번째 icon은 Immunity Debugger Python Shell입니다. Shell에서 계산을 하거나 간단한 script를 테스트 해볼 수 있습니다. [그림1-5]에 보이는 것처럼 저는 간단히 print문을 테스트 해보았습니다.

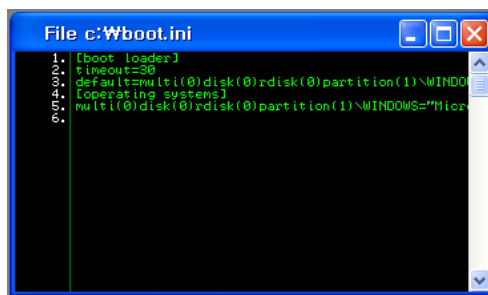


[그림1-6. Python Commands list]

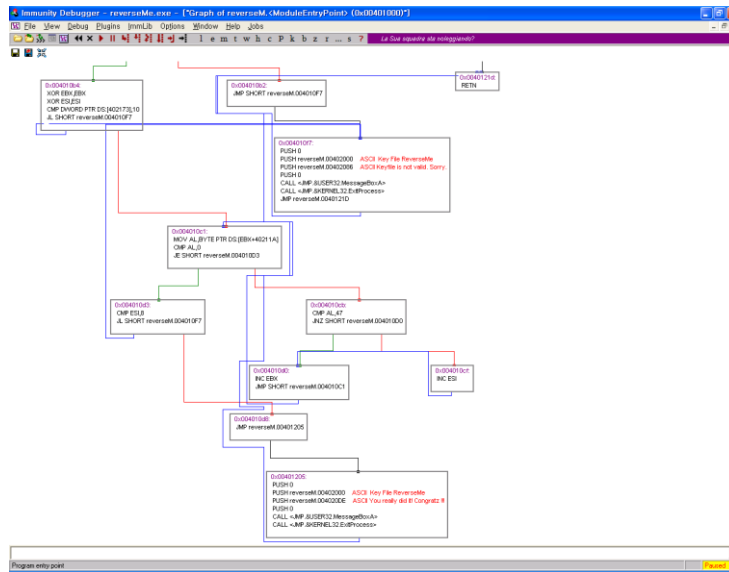
세 번째 icon은 python commands list입니다. [그림1-6]처럼 사용 할 수 있는 commands를 보여줍니다. 사용하고 싶은 PyCommand를 더블클릭 하게 되면 [그림1-7]처럼 보이는 창에 필요한 Arguments를 넣고 OK를 누르면 해당 Command가 실행됩니다. 예를 들어 openfile PyCommand를 더블클릭하고 PyCommand Arguments창에서 Arguments로 c:\boot.ini를 넣고 OK버튼을 눌러 실행해보겠습니다. 그러면 [그림1-8]같은 화면을 볼 수 있게 됩니다.



[그림1-7. PyCommand Arguments 창]



[그림1-8. Openfile PyCommand로 열린 boot.ini파일]

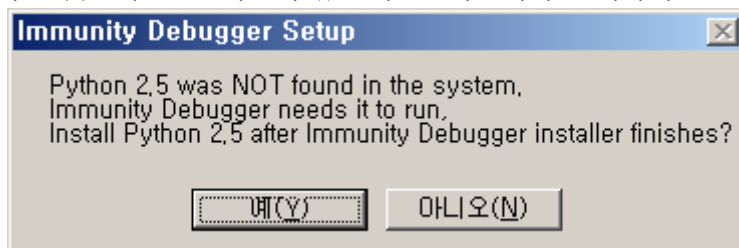


[그림1-9. 예제프로그램을 graph로 표현]

마지막 네 번째 icon은 graph입니다. [그림1-9]처럼 네 번째 icon을 누르면 Immunity Debugger가 현재 debugging하고 있는 파일을 간단하게 graph로 나타내줍니다. CPU창에서 disassembly된 code를 선택한 후 graph화 시키면 선택된 code부분부터 graph화 되어 화면에 나타나게 되는 것이 특징입니다. IDA처럼 해당 주소를 눌러도 이동하거나 하는 기능은 없습니다.

0x02. Python Script 1

Immunity Debugger의 가장 큰 특징이라고 할 수 있는 Python Script에 대해서 알아 보도록 하겠습니다. Immunity Debugger에서 Python Script를 실행시키기 위해선 Python이 설치 되어 있어야 합니다. 만약 Python이 설치 되지 않은 PC에 Immunity Debugger설치를 하게 되면 [그림2-1] Python을 같이 설치할 것인지 묻는 과정이 있으니 그 과정에서 설치하시면 됩니다.



[그림2-1. Python이 설치 되지 않은 상태에서 Immunity Debugger설치 시 뜨는 메시지]

내용을 진행하기 전에 Python에 대해서 한가지 꼭 알아 두셔야 할 점을 말씀 드리겠습니다. Python은 들여쓰기가 굉장히 중요합니다. Suite라고 불리는 것인데 이것을 지키지 않으면 문법적 오류가 생기므로 항상 주의하셔야 합니다. 특히 메모장과 같은 일반적인 Editor를 이용하실 때 오류가 발생한다면 들여쓰기를 확인해 보시는 게 좋겠습니다.

Python	C
<pre>if expression: if_suite</pre>	<pre>if (expression){ }</pre>
Ex) <pre>if (a > b): print "a is big"</pre>	Ex) <pre>if (a > b){ printf("a is big"); }</pre>

[표2-1. Python과 C의 Suite 비교]

일단 Immunity Debugger를 설치하면 함께 설치되는 예제 Python script를 분석해 보겠습니다.

```
import immllib

def main():
    imm = immllib.Debugger()
    pslist=imm.ps()
    for process in pslist:
        imm.Log("Process: %s - PID: %d" % (process[1],process[0]))

if __name__=="__main__":
    print "This module is for use within Immunity Debugger only"
```

[표2-2. py_example.py]

Line 1

```
import immllib
```

Immunity Debugger API가 들어 있는 module중 하나인 immllib를 사용하기 위해서 import문을 사용하였습니다. Module이라는 것은 class나 method들의 집합체라고 보시면 됩니다.

Line 3 ~ 7

```
def main():
    imm = immllib.Debugger()
    pslist = imm.ps()
    for process in pslist:
        imm.Log("Process: %s - PID: %d" % (process[1],process[0]))
```

main함수입니다. 함수는 다음과 같이 선언합니다.

```
def function_name(arguments):
    "optional documentation string"
    function_suite
```

Ex)

```
def main():
    print "Hello World!"
```

imm = immllib.Debugger()

immllib module에 들어있는 Debugger Class를 imm으로 줄여서 사용하겠다는 의미를 가집니다. Debugger Class에는 이름 그대로 Debugging을 하기 위한 많은 method들이 모여 있습니다. Immunity Debugger가 지원하는 module에 대해서 보다 자세한 정보를 원하시는 분들은 참고사이트 페이지에서 Immunity Debugger Online Documentation을 참고바랍니다.

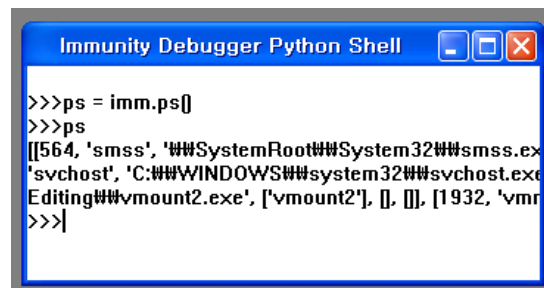
pslist = imm.ps()

[그림2-2]에서 보시는 것같이 Debugger Class의 ps method를 사용하여 실행중인 Process의 정보를 pslist에 리스트로 넣습니다. 리스트는 C의 배열이라고 생각하시면 편합니다. [그림2-3]을 보면 ps method를 이용하여 어떻게 Process의 정보를 얻을 수 있는지 알 수 있습니다.

Symbol: [imm.ps](#)
Likely type: [method Debugger.ps](#)
def Debugger.ps(self)
List all active processes.

@rtype: LIST @return: A list of tuples with process information (pid, name, path, services, tcp list, udp list)

[그림2-2. Debugger.ps의 정보]

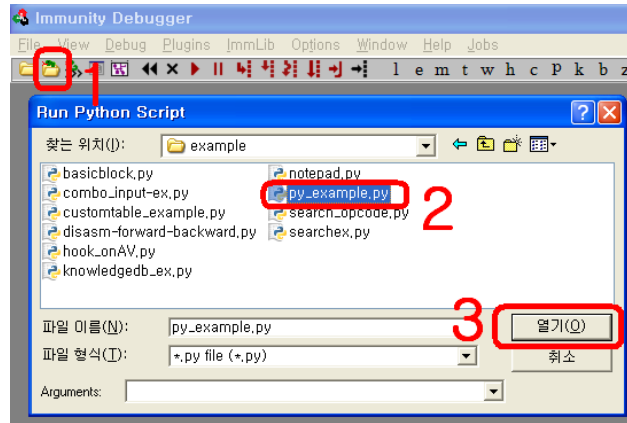


[그림2-3. imm.ps()]

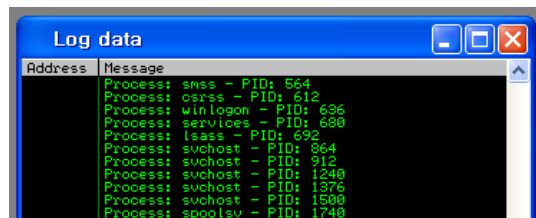
for process in pslist:

`imm.Log("Process: %s - PID: %d" % (process[1], process[0]))`

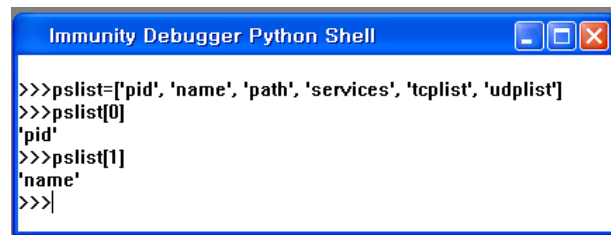
for문이긴 하지만 C의 for문과는 좀 다릅니다. Script 언어의 반복문인 foreach와 유사합니다. 실행 중인 모든 Process의 정보 중 name과 pid를 Immunity Debugger의 Log창에 남깁니다. [그림2-4]처럼 실행을 하게 되면 [그림2-5]처럼 결과를 볼 수 있게 됩니다.



[그림2-4. Python Script 실행하기]



[그림2-5. 실행 후 Log data에 남은 프로세스 정보]



[그림2-6. 출력될 정보의 위치]

Line 9 ~ 10

```
if __name__=="__main__":  
    print "This module is for use within Immunity Debugger only"
```

Shell에서 실행될 경우 사용자에게 메시지를 보여주기 위한 부분입니다.

이렇게 해서 간단하게 예제용으로 만들어진 Python Script파일을 분석을 완료하였습니다. 이제 조금 더 복잡한 Python Script파일을 분석해보겠습니다.

0x03. Python Script 2

```
"""
Lenal51 Tutorial 01,02
Olly + assembler + patching a basic reverseme

reverseMe.exe
"""

import immlib
import pefile

def main():
    imm = immlib.Debugger()
    currAddr = imm.getModule(imm.getDebuggedName()).getEntry()
    imm.Log("OEP: 0x%08x" % currAddr)

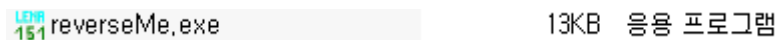
    goodBoy = 0x401218
    badBoys = 0x40107D,0x4010F7

    while(currAddr!=goodBoy):
        opcode = imm.disasm(imm.getRegs()['EIP'])
        if (opcode.isJump() or opcode.isConditionalJump()):
            """currAddr.jumpaddr=badBoys"""
            if opcode.jumpaddr==badBoys[0] or opcode.jumpaddr==badBoys[1]:
                size = opcode.getOpSize()
                nop = '\x90' * size
                imm.writeMemory(currAddr,nop)
                """currAddr.jumpaddr=goodBoy"""
            else:
                if opcode.isConditionalJump():
                    imm.writeMemory(currAddr,'\xEB')

        imm.stepOver()
        currAddr = imm.getRegs()['EIP']
    imm.stepOver()
    return 0

if __name__=="__main__":
    print "This module is for use within Immunity Debugger only"
```

[표3-1. 분석하고자 하는 Script]

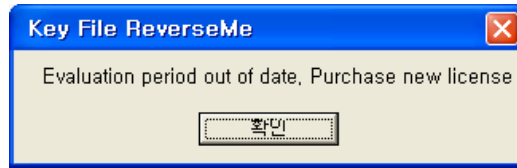
 13KB 응용 프로그램

[그림3-1. 예제프로그램1]

[표3-1]의 Script는 <http://forum.immunityinc.com/index.php?topic=141.0>에서 구할 수 있습니다. 그리고 [그림3-1]의 예제프로그램은 <http://www.tuts4you.com/download.php?view.122>에서 구할 수 있습니다. 이 Script는 굳이 Script를 만들지 않아도 Debugger만 있으면 충분히 가능한 부분을 Script로 만들어 자동화 시킨 것입니다.

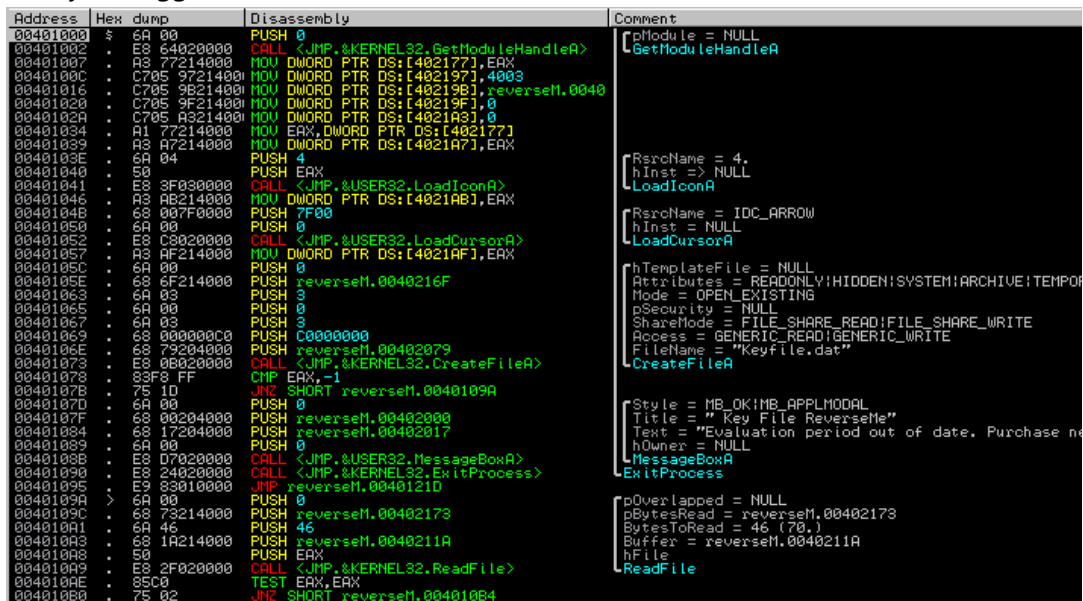
일단 우리가 공략해야 할 예제프로그램이 어떤 것인지 먼저 분석을 해본 후에 Script를 분석하도록 하겠습니다.

예제프로그램인 reverseMe.exe를 실행시켜 보았습니다. 그랬더니 다음과 같은 MessageBox를 볼 수 있었고 확인을 눌렀더니 프로그램이 종료 되었습니다.



[그림3-2. 평가기간이 끝났습니다.]

Immunity Debugger를 이용하여 예제프로그램을 열어보도록 하겠습니다.



[그림3-3. Disassembly된 예제프로그램]

ExitProcess위에 MessageBoxA를 보니 [그림3-2]에서 보았던 메시지가 있는 것을 알 수 있습니다. 그리고 그 위로는 CreateFileA API가 있고 아래로는 ReadFile API가 있습니다. 두 개의 API는 [표3-2], [표3-3]에서 확인 할 수 있습니다. [그림3-3]의 Comment들과 비교해서 보면 쉽게 알아볼 수 있습니다.

00401073에 BreakPoint를 설정하고 F9를 눌러서 실행시킨 후 00401078~0040107B에서 Keyfile.dat가 없을 때 분기하지 않는 것을 볼 수 있습니다. 0040107B에서 분기하지 않게 되면 [그림3-2]의 MessageBox를 보게 됩니다. 조건이 만족해 분기하게 되면 0040109A로 진행하게 됩니다.

```

BOOL ReadFile(
    HANDLE hFile, //읽고자 하는 파일의 핸들
    LPVOID lpBuffer, //읽는 데이터를 저장할 버퍼의 포인터
    DWORD nNumberOfBytesToRead, //읽고자 하는 바이트 수
    LPDWORD lpNumberOfBytesRead, //실제로 읽은 바이트 수를 리턴 받기 위한 출력용 인수
    LPOVERLAPPED lpOverlapped //비동기 입출력을 위한 OVERLAPPED 구조체의 포인터
);
    
```

[표3-2. ReadFile API]

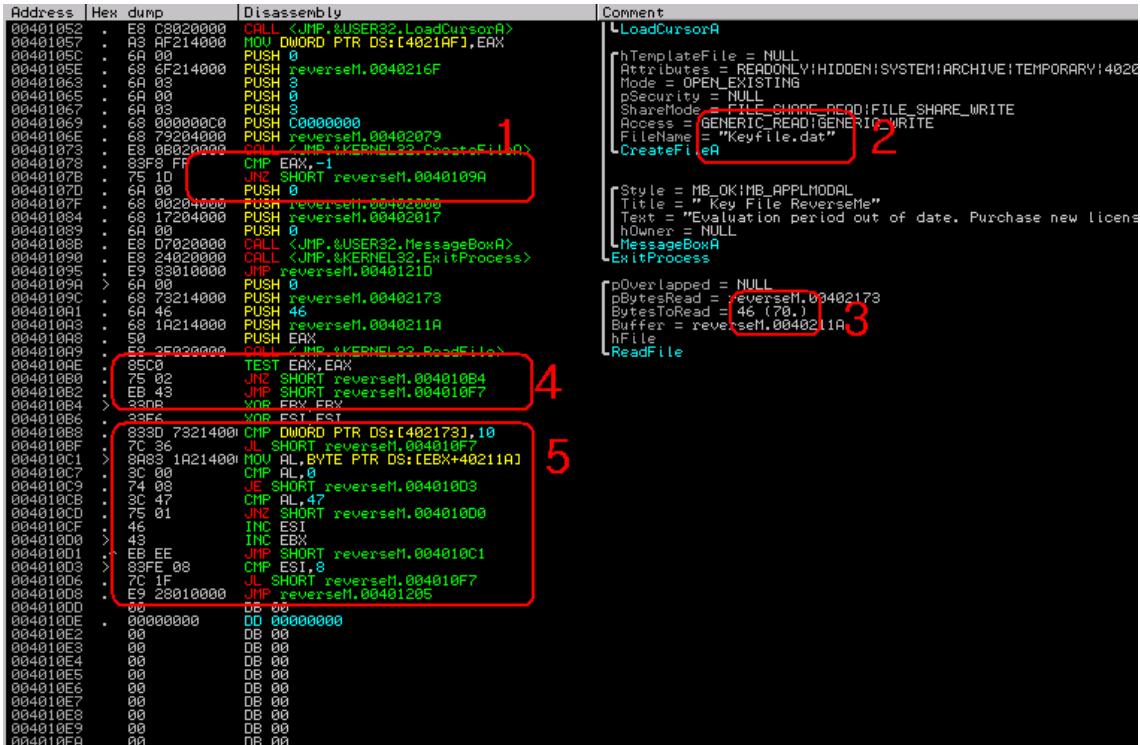
```

HANDLE CreateFile(
    LPCTSTR lpFileName, //열거나 만들고자 하는 파일의 완전경로를 문자열로 지정
    DWORD dwDesiredAccess, //파일에 대한 액세스 권한을 지정
    DWORD dwShareMode, //열려진 파일의 공유 모드를 지정
    LPSECURITY_ATTRIBUTES lpSecurityAttributes, //파일의 보안속성을 지정하는
SECURITY_ATTRIBUTES 구조체의 포인터
    DWORD dwCreationDisposition, //만들고자 하는 파일이 이미 있거나 또는 열고자 하는 파
일이 없을 경우의 처리를 지정
    DWORD dwFlagsAndAttributes, //파일의 속성과 여러 가지 옵션 설정
    HANDLE hTemplateFile //생성될 파일의 속성을 제공할 템플릿 파일
);

```

[표3-3. CreateFile API]

CreateFile API에서 확인 할 수 있는 것은 "Keyfile.dat"라는 파일이 필요하다는 것이고, ReadFile API에서 확인 할 수 있는 것은 데이터를 70Byte만큼만 읽어 온다는 것입니다. 아래 [그림3-4]를 보면서 자세히 알아 보도록 하겠습니다.



[그림3-4. Keyfile.dat의 조건]

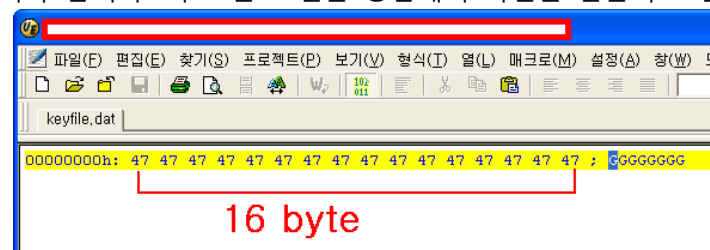
②에 보이는 FileName을 가진 파일이 존재하지 않을 경우 ①부분에서 0040109A로 분기하지 않게 되어 [그림3-2]와 같은 MessageBox를 보게 됩니다. 하지만 Keyfile.dat가 존재한다면 ③에 보이는 것처럼 데이터를 70Byte만큼 읽어 옵니다. 그런데 Keyfile.dat가 존재 하더라도 데이터가 아무것도 없다면 ④부분에서 004010F7로 분기하여 "Keyfile is not valid. Sorry"라는 MessageBox를 보게 되며, 데이터가 존재할 경우 ⑤부분에서 그 데이터를 가지고 간단한 확인 작업을 하게 됩니다. 그러면 ⑤부분을 보도록 하겠습니다.

```

004010B8      CMP DWORD PTR DS:[402173],10
//읽어 들인 Byte와 0x10(16)을 비교합니다.
004010BF      JL SHORT reverseM.004010F7
//읽어 들은 Byte의 크기가 0x10(16)보다 작다면 004010F7(실패 메시지)로 분기합니다.
004010C1      MOV AL, BYTE PTR DS:[EBX+40211A]
//AL에 버퍼의 내용을 1Byte 복사합니다.
004010C7      CMP AL, 0
//버퍼 값이 복사 되어 저장된 AL의 값과 0x00을 비교합니다.
004010C9      JE SHORT reverseM.004010D3
//같다면 004010D3(Loop를 빠져나가기 위한 첫 단계)으로 분기합니다.
004010CB      CMP AL, 47
//버퍼 값이 복사 되어 저장된 AL의 값과 0x47(G)을 비교합니다.
004010CD      JNZ SHORT reverseM.004010D0
//같지 않다면 004010D0로 분기합니다.
004010CF      INC ESI
//ESI Register를 1증가 시킵니다.
004010D0      INC EBX
//EBX Register를 1증가 시킵니다.
004010D1      JMP SHORT reverseM.004010C1
//004010C1으로 분기합니다. (Loop를 돌면서 버퍼의 값을 비교하며 연산하기 위함)
004010D3      CMP ESI, 8
//ESI Register의 값과 0x08(8)을 비교합니다.
004010D6      JL SHORT reverseM.004010F7
//ESI Register의 값이 0x08(8)보다 작다면 004010F7(실패 메시지)로 분기합니다.
004010D8      JMP reverseM.00401205
//00401205(성공 메시지)로 분기합니다.

```

004010D3~004010D8에서 보시는 것처럼 ESI Register의 값이 8이상 되어야 성공 메시지로 분기 할 수 있습니다. 즉 ESI Register를 증가 시켜주는 조건을 만족 시켜야 됩니다. ESI Register가 증가 되는 부분은 004010CF입니다. 004010CB에서 AL의 값과 0x47(G)를 비교하였을 때 같을 경우 진행되는 부분입니다. ESI Register의 값을 조건에 맞게 증가 시키기 위해서는 AL의 값이 연속적으로 8번 이상 0x47(G)이어야 합니다. 그리고 이 Loop를 빠져나가기 위해서는 004010C7에서 AL의 값이 한번 0x00(0)이어야 합니다. 이 모든 조건을 종합해서 파일을 만들어 보면 다음과 같습니다.



[그림3-5. Keyfile.dat 완성]

완성된 Keyfile.dat를 가지고 reverseMe.exe를 실행시켜 보았더니 다음과 같은 MessageBox를 볼 수 있었습니다.



[그림3-6. 성공메시지]

예제프로그램인 reverseMe.exe에 대해서 모두 알아 보았으니 이제 script에 대해서 알아 보도록 하겠습니다. 총 35Line이며 첫 줄부터 마지막 줄까지 한 부분씩 끊어서 설명하도록 하겠습니다.

Line 1 ~ 6

```
"""
Lenal51 Tutorial 01,02
Olly + assembler + patching a basic reverseme

reverseMe.exe
"""
```

Block Comment("""내용""")를 이용하여 주석을 달아 놓았습니다.

Line 8 ~ 9

```
import immlib
import pefile
```

C에서 헤더파일을 사용하기 위해서 #include를 사용하듯이 Python에서도 module을 사용하기 위해서는 import를 사용해야 합니다. import를 이용하여 immlib, pefile module을 사용하게끔 하였습니다. (pefile module은 <http://code.google.com/p/pefile>에서 구할 수 있습니다.)

Line 11 ~ 32

```
def main():
    imm = immlib.Debugger()
    currAddr = imm.getModule(imm.getDebuggedName()).getEntry()
    imm.Log("OEP: 0x%08x" % currAddr)

    goodBoy = 0x401218
    badBoys = 0x40107D,0x4010F7

    while(currAddr!=goodBoy):
        opcode = imm.disasm(imm.getRegs()['EIP'])
        if (opcode.isJump() or opcode.isConditionalJump()):
            """currAddr.jumpaddr=badBoys"""
            if opcode.jumpaddr==badBoys[0] or opcode.jumpaddr==badBoys[1]:
                size = opcode.getOpSize()
                nop = '\x90' * size
                imm.writeMemory(currAddr,nop)
                """currAddr.jumpaddr=goodBoy"""
            else:
                if opcode.isConditionalJump():
                    imm.writeMemory(currAddr,'\xEB')

        imm.stepOver()
        currAddr = imm.getRegs()['EIP']
    imm.stepOver()
    return 0
```

Main함수 입니다. Main함수 내의 code들을 부분적으로 살펴 보겠습니다.

Line 12 ~ 16

```
imm = immlib.Debugger()
currAddr = imm.getModule(imm.getDebuggedName()).getEntry()
imm.Log("OEP: 0x%08x" % currAddr)

goodBoy = 0x401218
badBoys = 0x40107D, 0x4010F7
```

imm = immlib.Debugger()

앞서 확인 했듯이 Debugging용 method를 사용하기 위해 import된 immlib의 Debugger Class를 imm으로 줄여서 선언하였습니다.

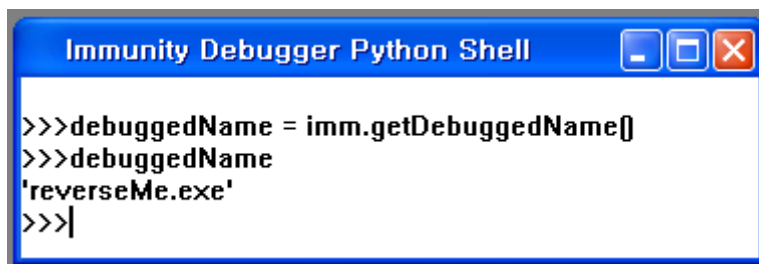
currAddr = imm.getModule(imm.getDebuggedName()).getEntry()

imm.getDebuggedName method로 현재 Debugging하고 있는 프로그램의 이름을 얻어오고, imm.getModule.getEntry method로 그 프로그램의 정보 중 Entry를 10진수로 가져옵니다.

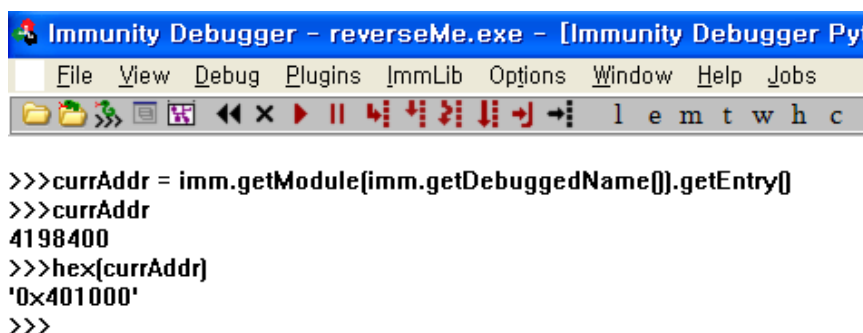
```
Symbol: imm.getDebuggedName
Likely type: method Debugger.getDebuggedName
def Debugger.getDebuggedName(self)
Get debugged name

@rtype: STRING
@return: Name of the Process been debugged
```

[그림3-7. getDebuggedName method]



[그림3-8. 이름 얻어 오기]



[그림3-9. Entry 확인하기]

imm.Log("OEP: 0x%08x" % currAddr)

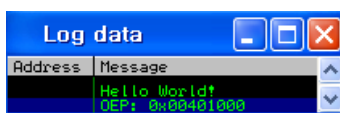
Immunity Debugger에 있는 Log Data창에 ("OEP: 0x%08x" % currAddr) 이라는 내용의 Log를 남깁니다. imm.Log("Hello World!")를 이용하면 Hello World!를 찍을 수 있습니다.


```

Immunity Debugger - reverseMe.exe - [Immunity Debugger Pyth
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P
>>>currAddr = imm.getModule(imm.getDebuggedName()).getEntry()
>>>imm.Log("OEP: 0x%08x" % currAddr)
0
>>>

```

[그림3-10. Entry를 Log data window로 출력하기]



[그림3-11. Log data window에 출력된 Entry]

imm.Log method에 대한 정보는 [그림3-12]과 같습니다. 그리고 [그림3-13]처럼 argument 값을 수정하여 Log data window에 text를 강조하여 나타낼 수 있습니다.

```

Symbol: imm.Log
Likely type: method Debugger.Log
def Debugger.Log(self, msg, address=0, highlight=False, gray=False, focus=0)
Adds a single line of ASCII text to the log window.

@type msg: STRING @param msg: Message (max size is 255 bytes)

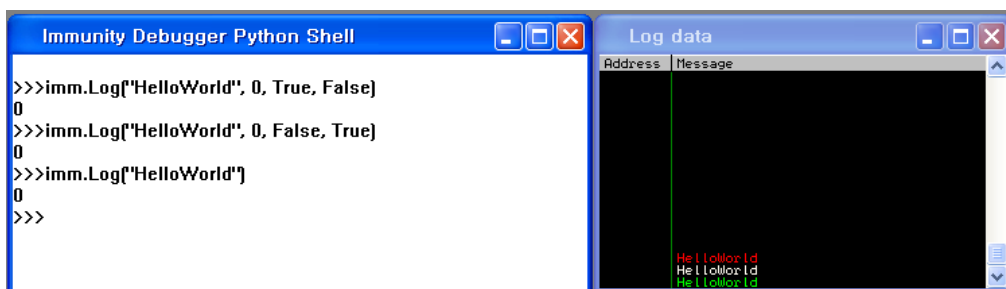
@type address: DWORD @param address: Address associated with the message

@type highlight: BOOLEAN @param highlight: Set highlight text

@type gray: BOOLEAN @param gray: Set gray text

```

[그림3-12. Log method]



[그림3-13. Text highlight & gray & normal]

goodBoy = 0x401218

badBoys = 0x40107D, 0x4010F7

goodBoy와 badBoys를 선언합니다. 변수에 들어가는 주소들이 어떤 내용을 가지고 있는지 확인해 보겠습니다.

0x401218 (goodBoy)

```
00401204 . 00 DB 00
00401205 > 6A 00 PUSH 0
00401207 . 68 00204000 PUSH reverseM.00402000
0040120C . 68 DE204000 PUSH reverseM.004020DE
00401211 . 6A 00 PUSH 0
00401213 . E8 4F010000 CALL <JMP.&USER32.MessageBoxA>
00401218 . E8 9C000000 CALL <JMP.&KERNEL32.ExitProcess>
0040121D > C3 RETN
0040121E . 00 DB 00
```

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

0x40107D (badBoys)

```
00401078 . 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 07020000 CALL <JMP.&USER32.MessageBoxA>
00401090 . E8 24020000 CALL <JMP.&KERNEL32.ExitProcess>
00401095 . E9 83010000 JMP reverseM.0040121D
```

```
Style = MB_OK!MB_APPLMODAL
Title = " Key File ReverseMe"
Text = "Evaluation period out of date. Purchase new license"
hOwner = NULL
MessageBoxA
ExitProcess
```

0x4010F7 (badBoys)

```
004010F4 . 00 DB 00
004010F5 . EB 00 JMP SHORT reverseM.004010F7
004010F7 > 6A 00 PUSH 0
004010F9 . 68 00204000 PUSH reverseM.00402000
004010FE . 68 86204000 PUSH reverseM.00402086
00401103 . 6A 00 PUSH 0
00401105 . E8 5D020000 CALL <JMP.&USER32.MessageBoxA>
0040110A . E8 AA010000 CALL <JMP.&KERNEL32.ExitProcess>
0040110F . E9 09010000 JMP reverseM.0040121D
00401114 . 00 DB 00
```

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

[표3-4. goodBoy, badBoys]

[표3-4]에 보이는 것과 같이 성공 메시지와 실패 메시지로 분류 되는걸 알 수 있습니다.

Line 19 ~ 32

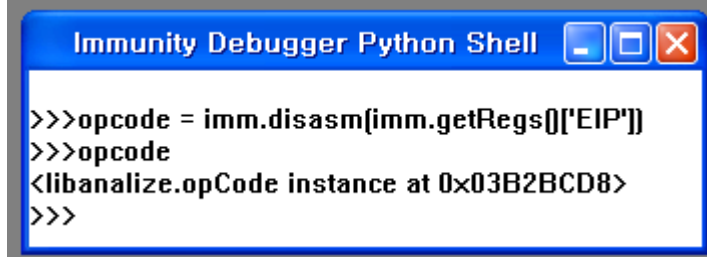
```
while (currAddr!=goodBoy) :
    opcode = imm.disasm(imm.getRegs() ['EIP'])
    if (opcode.isJump() or opcode.isConditionalJump()):
        ""currAddr.jmpaddr=badBoys""
        if opcode.jmpaddr==badBoys[0] or opcode.jmpaddr==badBoys[1]:
            size = opcode.getOpSize()
            nop = '\x90' * size
            imm.writeMemory(currAddr,nop)
            ""currAddr.jmpaddr=goodBoy""
        else:
            if opcode.isConditionalJump():
                imm.writeMemory(currAddr,'\xEB')
            imm.stepOver()
            currAddr = imm.getRegs() ['EIP']
    imm.stepOver()
return 0
```

while(currAddr != goodBoy):

이 script에서 가장 중요한 부분입니다. Loop문으로 while을 사용하였고 조건은 currAddr값이 goodBoy값과 같지 않을 때까지입니다.

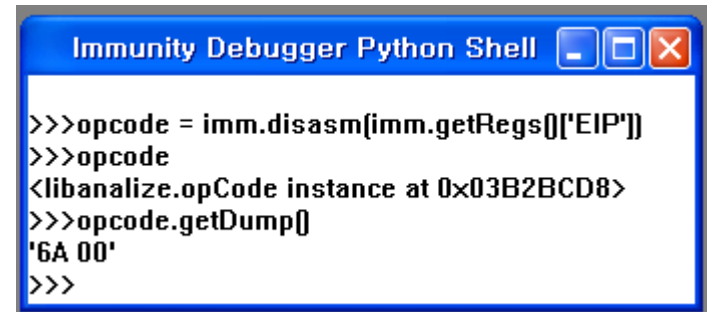
`opcode = imm.disasm(imm.getRegs()['EIP'])`

opcode에 EIP(Extended Instruction Pointer) Register가 가진 주소의 opcode를 넣습니다.



```
Immunity Debugger Python Shell
>>>opcode = imm.disasm(imm.getRegs()['EIP'])
>>>opcode
<libanalyze.opCode instance at 0x03B2BCD8>
>>>
```

opcode에 EB가 들어있는지 90이 있는지 알 턱이 없습니다.



```
Immunity Debugger Python Shell
>>>opcode = imm.disasm(imm.getRegs()['EIP'])
>>>opcode
<libanalyze.opCode instance at 0x03B2BCD8>
>>>opcode.getDump()
'6A 00'
>>>
```

getDump()를 이용해 실제 어떤 opcode가 들어 있는지 확인 할 수 있습니다.

[표3-5. opcode알아 내기]

if (opcode.isJmp() or opcode.isConditionalJmp()):

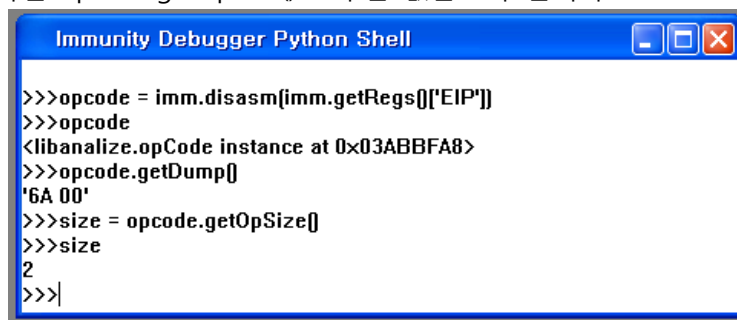
조건문 입니다. opcode가 isJmp이거나 isConditionalJmp일 때 조건이 성립하게 됩니다. isJmp는 JMP인 분기를 뜻하고, isConditionalJmp는 JE, JNZ와 같은 조건분기를 뜻합니다.

if opcode.jmpaddr == badBoys[0] or opcode.jmpadr == badBoys[1]:

이번에도 조건문 입니다. opcode가 분기이거나 조건분기 일 때 그 주소를 badBoys(0x40107D, 0x4010F7)와 비교하여 같을 때 조건이 성립합니다.

size = opcode.getOpSize()

size 변수에 byte단위로 조건이 성립한 opcode의 OpSize값을 넣습니다. 예를 들어 opcode에 '6A 00'이 들어 있다면 opcode.getOpSize()로 구한 값은 2가 됩니다.



```
Immunity Debugger Python Shell
>>>opcode = imm.disasm(imm.getRegs()['EIP'])
>>>opcode
<libanalyze.opCode instance at 0x03ABBFA8>
>>>opcode.getDump()
'6A 00'
>>>size = opcode.getOpSize()
>>>size
2
>>>
```

[그림3-14. Opcode size 구하기]

`nop = '\x90' * size`

nop변수에 얻어온 size만큼 \x90을 넣습니다. size가 2라면 nop변수에는 \x90가 두 번 들어가게 됩니다.

```
Immunity Debugger Python Shell
>>>size
2
>>>nop = '\x90' * size
>>>nop
'\x90\x90'
>>>nop = '\x90' * [size * 2]
>>>nop
'\x90\x90\x90\x90'
>>>|
```

[그림3-15. nop변수에 size만큼 \x90넣기]

imm.writeMemory(currAddr, nop)

currAddr이 가지고 있는 주소에 nop변수에 들어 있는 값인 \x90을 씁니다. 즉, badBoys가 가지고 있는 주소에 0x90(nop)을 써서 분기가 일어나지 않게 합니다. 아래 그림들을 보면 쉽게 이해할 수 있습니다. [그림3-16]은 writeMemory를 하기 전의 reverseMe.exe의 Entry입니다. "6A 00"을 가지고 있는 것을 확인할 수 있습니다. [그림3-17]은 Entry의 주소를 얻어온 후 2byte만큼 "90"을 writeMemory method를 이용하여 쓰는 과정입니다. [그림3-18]은 writeMemory가 끝난 후인데 "6A 00"이 "90 90"으로 변경된 것을 볼 수 있습니다.

Address	Hex	dump	Disassembly	Comment
00401000	6A	00	PUSH 0	
00401002	E8	64020000	CALL <JMP.&KERNEL32.GetModuleHandleA>	[pModule = NULL GetModuleHandleA
00401007	A3	77214000	MOV DWORD PTR DS:[402177],EAX	
0040100C	C705	9B214000	MOV DWORD PTR DS:[402197],4003	
00401016	C705	9B214000	MOV DWORD PTR DS:[40219B],reverseM.0040	
00401020	C705	9F214000	MOV DWORD PTR DS:[40219F],0	
0040102A	C705	A3214000	MOV DWORD PTR DS:[4021A3],0	

[그림3-16. Before writeMemory Entry]

```
Immunity Debugger Python Shell
>>>opcode = imm.getRegs()['EIP']
>>>hex(opcode)
'0x401000L'
>>>
>>>nop = '\x90' * 2
>>>nop
'\x90\x90'
>>>
>>>imm.writeMemory(opcode, nop)
2
>>>|
```

[그림3-17. nop쓰기]

Address	Hex	dump	Disassembly	Comment
00401000	90	90	NOP	[pModule
00401001	?	90	NOP	
00401002	E8	64020000	CALL <JMP.&KERNEL32.GetModuleHandleA>	[GetModuleHandleA
00401007	A3	77214000	MOV DWORD PTR DS:[402177],EAX	
0040100C	C705	9B214000	MOV DWORD PTR DS:[402197],4003	
00401016	C705	9B214000	MOV DWORD PTR DS:[40219B],reverseM.0040	
00401020	C705	9F214000	MOV DWORD PTR DS:[40219F],0	
0040102A	C705	A3214000	MOV DWORD PTR DS:[4021A3],0	

[그림3-18. After writeMemory Entry]

else:

if opcode.isConditionalJump():

imm.writeMemory(currAddr, 'WxEB')

두 번째 if문에 대한 else문 입니다. opcode가 분기나 조건분기가 맞지만 badBoys가 가지고 있는 값과 다를 경우에 opcode가 조건분기라면 currAddr이 가지고 있는 주소에 'WxEB'를 씁니다. ('WxEB'는 assembly로 표현하면 JMP가 됩니다. 즉, 조건분기일 경우에는 무조건 분기시키겠다는 뜻입니다.)

imm.stepOver()

조건문이 모두 끝나고 나서 stepOver로 한 라인을 진행시킵니다. [그림3-19]의 설명에는 주소를 정해주면 정해진 주소까지 진행한다고 되어 있지만 잘못된 설명입니다.

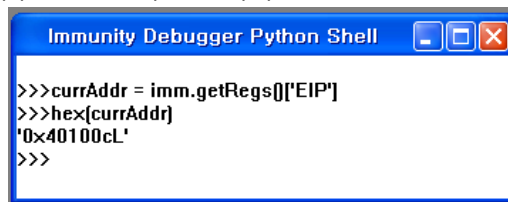
```
Symbol: imm_stepOver
Likely type: method Debugger.stepOver
def Debugger.stepOver(self, address=0)
Step-Over Process untill address.

@type address: DWORD
@param address: (Optional, Def= 0) Address
```

[그림3-19. stepOver method]

currAddr = imm.getRegs()['EIP']

currAddr에 현재 Registers값 중 EIP값을 넣습니다. Loop를 진행시키는데 있어서 진행 중인 위치의 주소를 가지고 오는 매우 중요한 부분입니다.



```
Immunity Debugger Python Shell
>>>currAddr = imm.getRegs()['EIP']
>>>hex[currAddr]
'0x40100cL'
>>>
```

[그림3-20. EIP값 가지고 오기]

imm.stepOver()

currAddr의 값이 goodBoy와 같아져 Loop문이 끝나면 stepOver로 한 라인을 더 진행시킵니다.

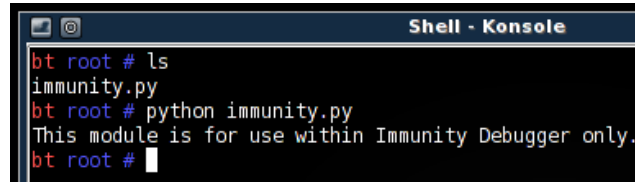
return 0

Main함수에 0을 리턴 해 프로그램을 종료합니다.

Line 34 ~ 35

```
if __name__=="__main__":
    print "This module is for use within Immunity Debugger only"
```

[그림3-21]처럼 shell에서 실행될 경우 사용자에게 메시지를 보여주기 위한 부분입니다.

A screenshot of a terminal window titled "Shell - Konsole". The terminal shows the following commands and output:

```
bt root # ls
immunity.py
bt root # python immunity.py
This module is for use within Immunity Debugger only.
bt root #
```

[그림3-21. shell에서 실행 했을경우]

reverseMe.exe를 Immunity Debugger에서 열고 이 script를 실행하게 되면 자동으로 한 라인씩 stepOver로 진행시키며, 정해놓은 조건에 따라서 메모리를 자동으로 수정하여 성공메시지까지 진행하게 됩니다. 이렇게 해서 예제프로그램과 script의 분석을 모두 마쳤습니다. Part2에서도 part1과 마찬가지로 script분석을 할 것입니다. Part2에서 보다 다양한 기능들에 대해서 다루도록 하겠습니다.

0x04. 참고사이트 & 참고문헌

- **Immunity Debugger** – <http://www.immunityinc.com/>
Immunity Debugger 공식사이트
- **Immunity Debugger Forum** – <http://forum.immunityinc.com/>
Immunity Debugger 공식포럼
- **Immunity Debugger Online Documentation** –
<http://debugger.immunityinc.com/updata/Documentation/ref/>
Immunity Debugger API 온라인 문서
- **CORE 파이썬 프로그래밍** – Wesley J. Chun, 백종현 외 공역
Python 프로그래밍 서적
- **Olly Debugger** – <http://www.olldb.de/>
Immunity Debugger의 모체인 Olly Debugger
- **Windows API 연구사이트** – <http://www.winapi.co.kr>
윈도우 API에 관한 많은 내용이 있음
- **Windows API 정복** – 김상형 저
Windows API 서적
- **pefile module** – <http://code.google.com/p/pefile>
pefile module 페이지
- **Lena151 tutorial 01, 02 Script** - <http://forum.immunityinc.com/index.php?topic=141.0>
본 문서에서 사용하는 script
- **reverseMe.exe** - <http://www.tuts4you.com/download.php?view.122>
본 문서에서 사용하는 예제프로그램