



BEISTLAB FOR SECURITY SINCE 2001

# [Immunity Debugger & Python (Part 2)]

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

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

# Abstract

본 문서에서는 Immunity Debugger와 쉽고 강력한 프로그래밍 언어 중 하나인 Python을 다루고 있습니다. 이번 문서에는 Part 1 에서 다루지 않았던 Immunity Debugger Forum의 f3 님이 만드신 나머지 3 개의 Python Script를 분석하겠습니다. 그리고 Immunity Debugger에 포함된 유용한 PyCommand를 분석하여 Immunity Debugger에 대해서 더 알아보도록 하겠습니다. 본 문서를 어려움 없이 읽기 위해서는 기본적인 어셈블리지식과 Python에 대해서 기초적인 문법 정도는 알고 있어야 합니다.

# Contents

0x01. Python Script 1

0x02. Python Script 2

0x03. Python Script 3

0x04. 유용한 PyCommand

0x05. 참고사이트 & 참고문헌

## 0x01. Python Script 1

```
"""
Lenal51 Tutorial 03
Basic nag removal + header problems

RegisterMe.exe
"""

import immlib
import pefile

def main():
    imm = immlib.Debugger()
    path = imm.getModule(imm.getDebuggedName()).getPath()
    pe = pefile.PE(path)
    """
    changing the entry point to 00401024 will
    skip the nag too. We can do that because
    the code before the nag is not important.
    """
    pe.OPTIONAL_HEADER.AddressOfEntryPoint=0x1024
    pe.write(filename='RegisterMe1.exe')

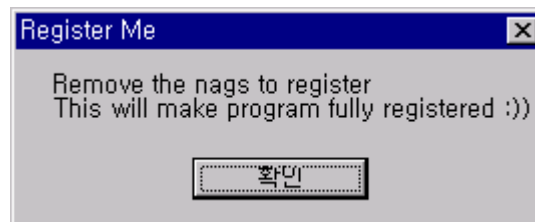
if __name__=="__main__":
    print "This module is for use within Immunity Debugger only"
```

[표 1-1. 분석하고자 하는 Script]

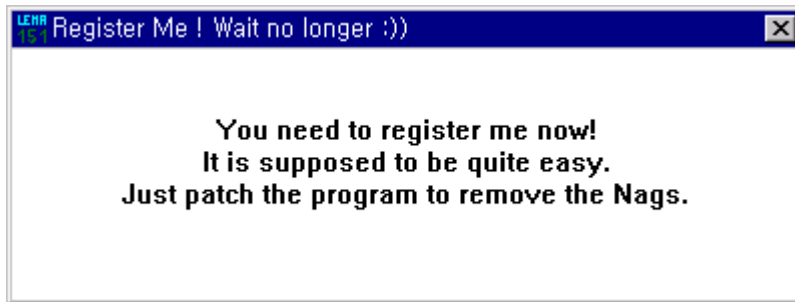
 RegisterMe.exe                      5KB 응용 프로그램

[그림 1-1. 예제프로그램]

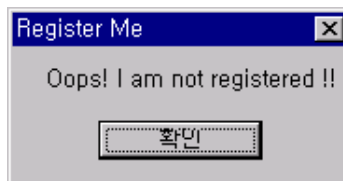
[표 1-1]의 Script 는 <http://forum.immunityinc.com/index.php?topic=141.0> 에서 구할 수 있습니다. 그리고 [그림 1-1]의 예제프로그램은 <http://www.tuts4you.com/download.php?view.124> 에서 구할 수 있습니다. 분석하고자 하는 Script 에 대해서 간단히 소개 하자면 프로그램의 시작지점(Entry Point)을 수정한 후 'RegisterMe1.exe'라는 파일명으로 저장까지 하는 Script 입니다. 일단 Script 를 분석하기 전에 예제프로그램을 먼저 분석해보도록 하겠습니다.



[그림 1-2. 실행 후 뜨는 창]

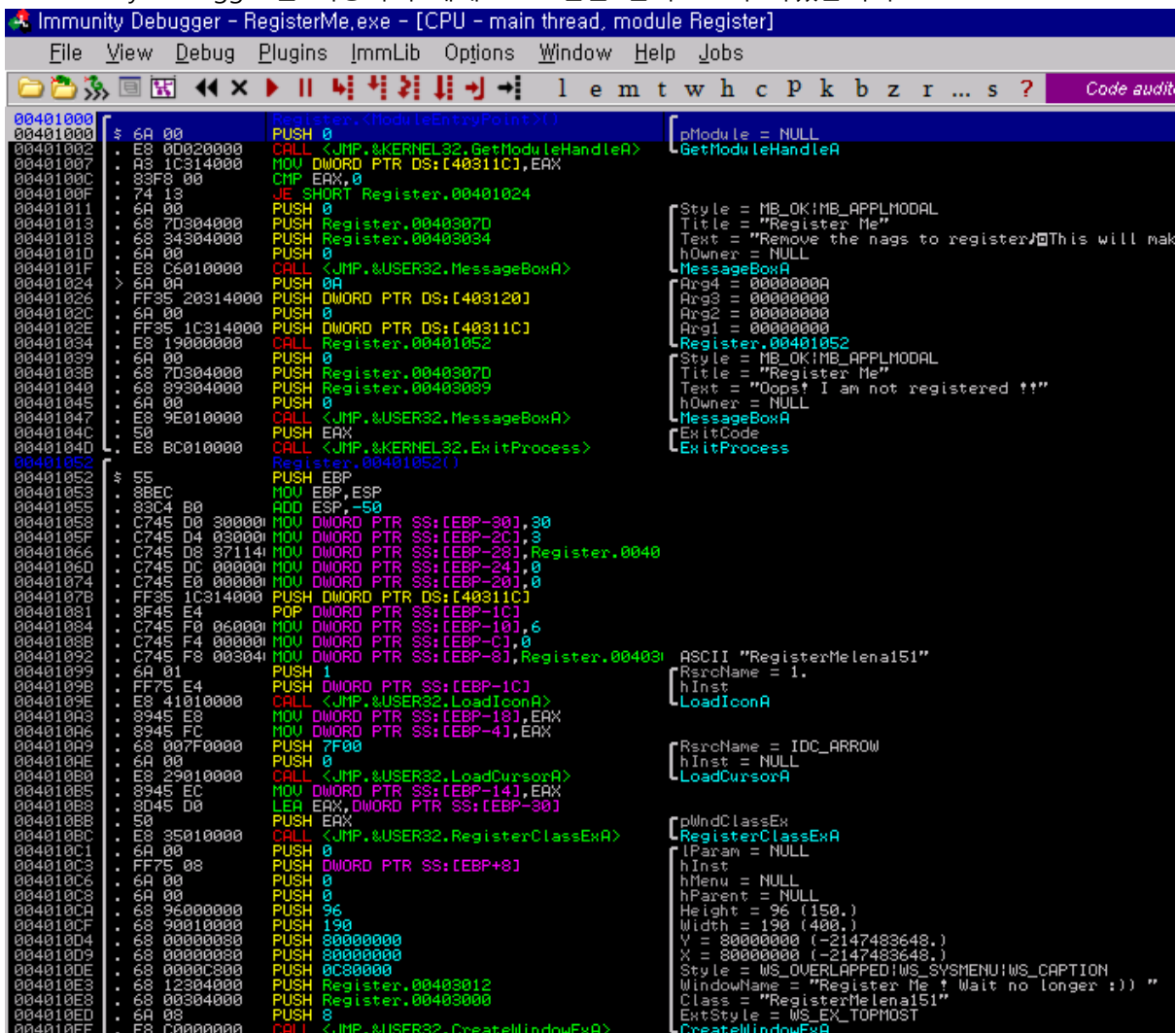


[그림 1-3. 확인을 누른 후 뜨는 창]



[그림 1-4. 종료할 경우 뜨는 창]

Immunity Debugger 를 이용하여 예제프로그램을 열어보도록 하겠습니다.



[그림 1-5. Disassembly 된 예제프로그램]

0x0040101F 를 보면 MessageBoxA API 가 호출되는데 그 내용을 보면 [그림 1-2]와 내용이 동일한 것을 알 수 있습니다. [그림 1-2]가 요구하는 데로 저 부분이 실행이 되지 않게 해야 합니다.

MessageBoxA API 가 호출되는 부분을 0x90(NOP)으로 수정하여 호출되지 않게 할 수 있지만, 이 script 에서는 다른 방법으로 MessageBoxA API 가 호출되지 않게 하고 있습니다. 그건 바로 프로그램의 시작지점(Entry Point)을 수정하는 방법입니다. 이제 Script 를 분석할 텐데 Script 는 총 19 줄이며 Part 1 에서 설명했던 Block Comment 와 module import 는 생략하도록 하겠습니다.

Line 11 ~ 16

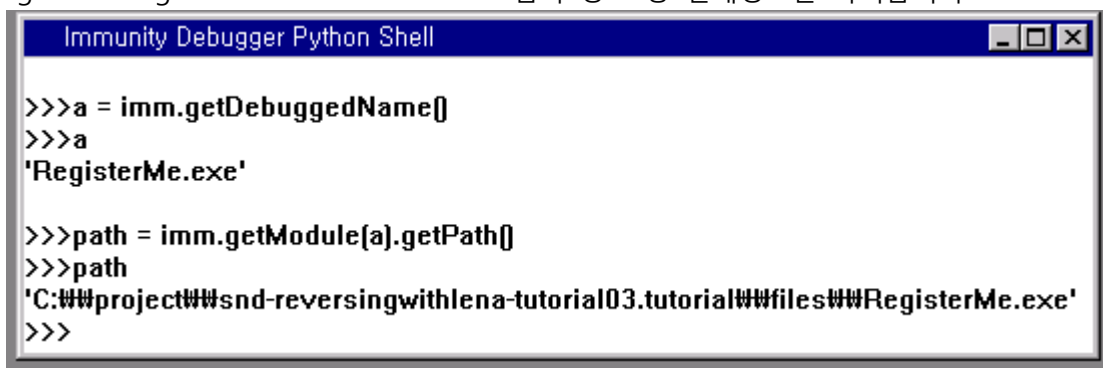
```
def main():
    imm = immlib.Debugger()
    path = imm.getModule(imm.getDebuggedName()).getPath()
    pe = pefile.PE(path)
    pe.OPTIONAL_HEADER.AddressOfEntryPoint=0x1024
    pe.write(filename='RegisterMe1.exe')
```

### **imm = immlib.Debugger()**

immlib module에 있는 Debugger Class를 imm으로 줄여서 사용하겠다는 의미를 가집니다.

### **path = imm.getModule(imm.getDebuggedName()).getPath()**

imm.getDebuggedName method로 현재 Debugging하고 있는 프로그램의 이름을 얻어오고, imm.getModule.getPath method로 그 프로그램의 정보 중 절대경로를 가져옵니다.



```
Immunity Debugger Python Shell
>>>a = imm.getDebuggedName()
>>>a
'RegisterMe.exe'

>>>path = imm.getModule[a].getPath()
>>>path
'C:\project\#snd-reversingwithlena-tutorial03.tutorial\files\RegisterMe.exe'
>>>
```

[그림1-6. 파일이름과 절대경로 얻어오기]

### **pe = pefile.PE(path)**

pe라는 변수에 앞서 얻어온 파일의 절대경로를 이용하여 그 파일의 PE정보를 얻어옵니다. PE(Portable Executable)는 Win32의 기본적인 파일형식입니다. 우리가 사용하는 exe, dll 과 같은 윈도우 플랫폼에서 실행 가능한 파일들을 말합니다. 그런 파일들에는 일정한 형식이 있습니다. 우리는 그것을 PE파일형식(Portable Executable File Format)이라고 부릅니다.

### **pe.OPTIONAL\_HEADER.AddressOfEntryPoint=0x1024**

얻어온 PE정보 중 OPTIONAL\_HEADER부분의 AddressOfEntryPoint를 0x1024로 수정합니다.

```

Immunity Debugger Python Shell
>>>a = pe.OPTIONAL_HEADER.AddressOfEntryPoint
>>>a
4096
>>>hex(a)
'0x1000'
>>>

```

[그림1-7. 수정하기 전의 AddressOfEntryPoint 값]

```

Immunity Debugger - RegisterMe.exe - [CPU - main thread, module Register]
File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z
00401000 Register.<ModuleEntryPoint>()
00401000 $ 6A 00 PUSH 0
00401002 . E8 00020000 CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007 . A3 1C314000 MOV DWORD PTR DS:[40311C],EAX
0040100C . 83F8 00 CMP EAX,0
0040100F . 74 13 JE SHORT Register.00401024
00401011 . 6A 00 PUSH 0
00401013 . 68 7D304000 PUSH Register.0040307D
00401018 . 68 34304000 PUSH Register.00403034
0040101D . 6A 00 PUSH 0
0040101F . E8 C6010000 CALL <JMP.&USER32.MessageBoxA>
00401024 Register.<ModuleEntryPoint>()
00401024 > 6A 0A PUSH 0A
00401026 . FF35 20314000 PUSH DWORD PTR DS:[403120]
0040102C . 6A 00 PUSH 0
0040102E . FF35 1C314000 PUSH DWORD PTR DS:[40311C]
00401034 . E8 19000000 CALL Register.00401052
00401039 . 6A 00 PUSH 0
0040103B . 68 7D304000 PUSH Register.0040307D
00401040 . 68 89304000 PUSH Register.00403089
00401045 . 6A 00 PUSH 0
00401047 . E8 9E010000 CALL <JMP.&USER32.MessageBoxA>
0040104C . 50 PUSH EAX
0040104D . E8 BC010000 CALL <JMP.&KERNEL32.ExitProcess>

```

[그림1-8. 변경 전의 AddressOfEntryPoint]

```

00401000 Register.<ModuleEntryPoint>()
00401000 6A 6A DB 6A CHAR 'j'
00401001 00 DB 00
00401002 E8 DB E8
00401003 00 DB 00
00401004 02 DB 02
00401005 00 DB 00
00401006 00 DB 00
00401007 A3 DB A3
00401008 1C314000 DD Register.0040311C
0040100C 83 DB 83
0040100D F8 DB F8
0040100E 00 DB 00
0040100F 74 DB 74 CHAR 't'
00401010 13 DB 13
00401011 6A DB 6A CHAR 'j'
00401012 00 DB 00
00401013 . 68 7D 30 40 00 ASCII "h)00",0
00401018 . 68 34 30 40 00 ASCII "h400",0
0040101D 6A DB 6A CHAR 'j'
0040101E 00 DB 00
0040101F E8 DB E8
00401020 C6 DB C6
00401021 01 DB 01
00401022 00 DB 00
00401023 00 DB 00
00401024 Register.<ModuleEntryPoint>()
00401024 $ 6A 0A PUSH 0A
00401026 . FF35 20314000 PUSH DWORD PTR DS:[403120]
0040102C . 6A 00 PUSH 0
0040102E . FF35 1C314000 PUSH DWORD PTR DS:[40311C]
00401034 . E8 19000000 CALL Register.00401052
00401039 . 6A 00 PUSH 0
0040103B . 68 7D304000 PUSH Register.0040307D
00401040 . 68 89304000 PUSH Register.00403089
00401045 . 6A 00 PUSH 0
00401047 . E8 9E010000 CALL <JMP.&USER32.MessageBoxA>
0040104C . 50 PUSH EAX
0040104D . E8 BC010000 CALL <JMP.&KERNEL32.ExitProcess>

```

[그림1-9. 변경 후의 AddressOfEntryPoint]

HEADERS INFO			
Address of Entry Point: <input type="text" value="00401000"/>		Real Image Checksum: <input type="text" value="0000D15Eh"/>	
Field Name	Data Value	Description	
Machine	014Ch	i386	
Number of Sections	0004h		
Time Date Stamp	38D1291Eh	16/03/2000 18:34:06	
Pointer to Symbol Table	00000000h		
Number of Symbols	00000000h		
Size of Optional Header	00E0h		
Characteristics	010Fh		
Magic	0108h	PE32	
Linker Version	0C05h	5.12	
Size of Code	00000400h		
Size of Initialized Data	00000400h		
Size of Uninitialized Data	00000000h		
Address of Entry Point	00401000h		
Base of Code	00001000h		
Base of Data	00002000h		
Image Base	00400000h		

[그림1-10. PE Explorer로 확인한 변경 전 RegisterMe.exe의 AddressOfEntryPoint 값]

AddressOfEntryPoint는 PE파일 실행 시 실행될 첫 번째 명령의 RVA(Relative Virtual Address)를 말합니다. RVA는 상대적 가상 주소라고 부르며 offset이라고 생각하면 됩니다. offset이 존재하려면 기준이 되어주는 값이 있어야 하는데 PE에선 이것을 [그림1-9]의 마지막에 보이는 ImageBase라고 부릅니다. ImageBase는 PE파일이 메모리에 맵핑 될 시작 주소를 말합니다. 대개 exe파일의 경우에는 0x00400000의 값을 갖고 dll은 0x10000000의 값을 갖습니다. RegisterMe.exe의 ImageBase는 0x00400000이고 RVA는 0x00001000이므로 이 두 값을 더한 0x00401000이 EP(EntryPoint)가 되는 것입니다.

따라서 이 코드는 RegisterMe.exe가 실행 시 실행될 첫 번째 명령이 있는 RVA를 0x1000에서 0x1024로 변경하는 것을 의미합니다. 그 결과로 [그림1-9]처럼 MessageBoxA API바로 아래가 처음 실행될 명령이 되므로 MessageBox API가 호출되지 않게 됩니다.

#### pe.write(filename = 'RegisterMe1.exe')

AddressOfEntryPoint가 0x1024로 수정된 RegisterMe.exe를 RegisterMe1.exe로 저장합니다. 저장되는 기본위치는 Immunity Debugger가 설치된 디렉터리 입니다. 다른 곳으로 변경을 원한다면 절대경로로 파일명까지 지정해주시면 됩니다. 이렇게 해서 생성된 RegisterMe1.exe를 실행하게 되면 [그림1-2]의 창은 뜨지 않고 바로 [그림1-3]의 창이 나타나게 됩니다.



## 0x02. Python Script 2

Python Script 1에서 생성한 RegisterMe1.exe를 이용하여 다른 Python Script 2를 진행하도록 하겠습니다.

```
import immlib

def main():
    imm = immlib.Debugger()
    """Killing the nag ?
    The solution is easy though : just
    NOP the call to the messagebox.
    But let's do it properly and NOP
    The arguments too. Like this ..."""
    for nop in range(0x401039, 0x40104C):
        imm.writeMemory(nop, '\x90')

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

[표 2-1. 분석하고자 하는 Script]

Line 5 ~ 6

```
def main():
    imm = immlib.Debugger()
    for nop in range(0x401039, 0x40104C):
        imm.writeMemory(nop, '\x90')
```

### **imm = immlib.Debugger()**

immlib module에 있는 Debugger Class를 imm으로 줄여서 사용하겠다는 의미를 가집니다.

### **for nop in range(0x401039, 0x40104C):**

#### **imm.writeMemory(nop, '\x90')**

for문, range함수 그리고 imm.writeMemory method를 사용하여 [그림1-9]에 보이는 0x401039부터 0x40104C전까지 nop code(0x90)를 메모리에 씁니다.

Symbol: [range](#)

Likely type: [builtin function builtin.range](#)

**def range(start=None, stop=None, step=None)**  
range([start,] stop[, step]) -> list of integers

Return a list containing an arithmetic progression of integers. range(i, j) returns [i, i+1, i+2, ..., j-1]; start (i) defaults to 0. When step is given, it specifies the increment (or decrement). For example, range(4) returns [0, 1, 2, 3]. The end point is omitted! These are exactly the valid indices for a list of 4 elements.

[그림2-1. range함수]

Symbol: [imm.writeMemory](#)  
 Likely type: [method Debugger.writeMemory](#)  
**def Debugger.writeMemory**(self, address, buf)  
 Write buffer to memory address.

@type address: DWORD  
 @param address: Address

@type buf: BUFFER  
 @param buf: Buffer

[그림2-2. Imm.writeMemory method]

```

00401024 Register.<ModuleEntryPoint>()
00401024 Register.<ModuleEntryPoint>()
00401024 $ 6A 0A PUSH 0A Arg4 = 0000000A
00401026 . FF35 20314000 PUSH DWORD PTR DS:[403120] Arg3 = 00000000
0040102C . 6A 00 PUSH 0 Arg2 = 00000000
0040102E . FF35 1C314000 PUSH DWORD PTR DS:[40311C] Arg1 = 00000000
00401034 . E8 19000000 CALL Register.00401052 Register.00401052
00401039 . 6A 00 PUSH 0 Style = MB_OK!MB_APPLMODAL
0040103B . 68 7D304000 PUSH Register.0040307D Title = "Register Me"
00401040 . 68 89304000 PUSH Register.00403069 Text = "Oops! I am not registered !!"
00401045 . 6A 00 PUSH 0 hOwner = NULL
00401047 . E8 9E010000 CALL <JMP.&USER32.MessageBoxA> MessageBoxA
0040104C . 50 PUSH EAX ExitCode
0040104D . E8 BC010000 CALL <JMP.&KERNEL32.ExitProcess> ExitProcess
  
```

[그림2-3. Script 실행 전 0x401039 ~ 0x40104C]

```

00401024 Register.<ModuleEntryPoint>()
00401024 Register.<ModuleEntryPoint>()
00401024 $ 6A 0A PUSH 0A Arg4 = 0000000A
00401026 . FF35 20314000 PUSH DWORD PTR DS:[403120] Arg3 = 00000000
0040102C . 6A 00 PUSH 0 Arg2 = 00000000
0040102E . FF35 1C314000 PUSH DWORD PTR DS:[40311C] Arg1 = 00000000
00401034 . E8 19000000 CALL Register.00401052 Register.00401052
00401039 . 90 NOP
0040103A . 90 NOP
0040103B . 90 NOP
0040103C . 90 NOP
0040103D . 90 NOP
0040103E . 90 NOP
0040103F . 90 NOP
00401040 . 90 NOP
00401041 . 90 NOP
00401042 . 90 NOP
00401043 . 90 NOP
00401044 . 90 NOP
00401045 . 90 NOP
00401046 . 90 NOP
00401047 . 90 NOP
00401048 . 90 NOP
00401049 . 90 NOP
0040104A . 90 NOP
0040104B . 90 NOP
0040104C . 50 PUSH EAX ExitCode
0040104D . E8 BC010000 CALL <JMP.&KERNEL32.ExitProcess> ExitProcess
  
```

[그림2-4. Script 실행 후 0x401039 ~ 0x40104C]

## 0x03. Python Script 3

이번에는 새로운 Script와 예제프로그램 RegisterMe.Oops.exe를 이용하여 진행하도록 하겠습니다. Script내용은 다음과 같습니다.

```
"""
Lenal51 Tutorial 03
Basic nag removal + header problems

RegisterMe.Oops.exe
"""
import immlib
import pefile

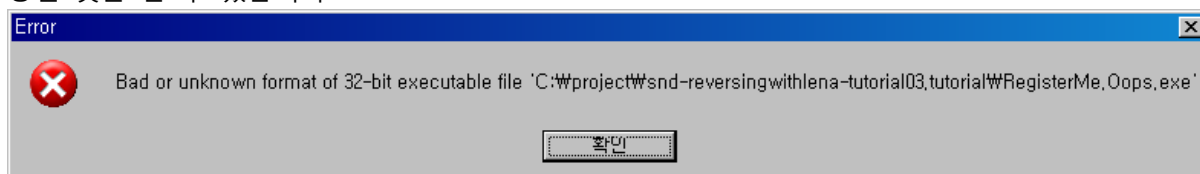
def main():
    imm = immlib.Debugger()
    module = imm.getModule(imm.getDebuggedName())
    pe = pefile.PE(module.getPath())
    """some virii but also protectors
    (see later tuts, I'll come back to this)
    Can deliberately manipulate data in the
    PE header as anti-debug tricks (...etc)."""
    pe.OPTIONAL_HEADER.SizeOfCode = 0x400
    pe.OPTIONAL_HEADER.SizeOfInitializedData = 0xA00
    pe.OPTIONAL_HEADER.BaseOfCode = 0x1000
    pe.OPTIONAL_HEADER.BaseOfData = 0x2000
    pe.OPTIONAL_HEADER.NumberOfRvaAndSizes = 0x10
    """Export Table address"""
    pe.OPTIONAL_HEADER.DATA_DIRECTORY[0].VirtualAddress = 0
    """Export Table size"""
    pe.OPTIONAL_HEADER.DATA_DIRECTORY[0].Size = 0
    pe.write('RegisterMe.Oops.1.exe')

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

[표3-1. 분석하고자 하는 Script]

[표3-1]의 Block Comment에 나와 있듯이 몇몇 virii와 protector들이 PE header를 고의로 조작하여 anti-debug tricks을 사용한다고 합니다. 예제프로그램인 RegisterMe.Oops.exe를 Immunity Debugger를 사용해 열게 되면 [그림3-1]과 같은 오류 메시지를 확인 할 수 있습니다. 하지만 예제프로그램을 직접 실행하게 되면 아무런 이상 없이 실행이 됩니다. 왜냐하면 윈도우는 조작된 PE data를 적절하게 무시하고 프로그램을 어떻게든 실행하지만, Immunity Debugger와 같은 Debugger들은 보다 엄격하게 PE를 확인하기 때문입니다.

그리고 이 파일을 직접실행 해 보시면 Python Script 1에서 사용했던 예제프로그램과 동일한 내용인 것을 알 수 있습니다.



[그림3-1. PE header 조작으로 인하여 발생한 Immunity Debugger의 Error MessageBox]

PE Header가 고의로 조작이 있는 것을 알았으니 PE Explorer라는 툴을 사용하여, 두 예제프로그램의 PE Header 정보를 비교해서 조작된 부분을 찾아보도록 하겠습니다.

HEADERS INFO						
Address of Entry Point: 00401000		Real Image Checksum: 000111A8h				
Field Name	Data Value	Description		Field Name	Data Value	Description
Machine	014Ch	i386		Section Alignment	00001000h	
Number of Sections	0004h			File Alignment	00000200h	
Time Date Stamp	38D1291Eh	16/03/2000 18:34:06		Operating System Version	00000004h	4.0
Pointer to Symbol Table	00000000h			Image Version	00000000h	0.0
Number of Symbols	00000000h			Subsystem Version	00000004h	4.0
Size of Optional Header	00E0h			Win32 Version Value	00000000h	Reserved
Characteristics	010Fh			Size of Image	00005000h	20480 bytes
Magic	0108h	PE32		Size of Headers	00000400h	
Linker Version	0C05h	5.12		Checksum	0000B499h	
Size of Code	40000400h			Subsystem	0002h	Win32 GUI
Size of Initialized Data	40000A00h			Dll Characteristics	0000h	
Size of Uninitialized Data	00000000h			Size of Stack Reserve	00100000h	
Address of Entry Point	00401000h			Size of Stack Commit	00001000h	
Base of Code	40001000h			Size of Heap Reserve	00100000h	
Base of Data	40002000h			Size of Heap Commit	00001000h	
Image Base	00400000h			Loader Flags	00000000h	Obsolete
				Number of Data Directories	40000004h	

[그림3-2. RegisterMe.Oops.exe의 PE Header 정보 1]

HEADERS INFO						
Address of Entry Point: 00401000		Real Image Checksum: 0000D15Eh				
Field Name	Data Value	Description		Field Name	Data Value	Description
Machine	014Ch	i386		Section Alignment	00001000h	
Number of Sections	0004h			File Alignment	00000200h	
Time Date Stamp	38D1291Eh	16/03/2000 18:34:06		Operating System Version	00000004h	4.0
Pointer to Symbol Table	00000000h			Image Version	00000000h	0.0
Number of Symbols	00000000h			Subsystem Version	00000004h	4.0
Size of Optional Header	00E0h			Win32 Version Value	00000000h	Reserved
Characteristics	010Fh			Size of Image	00005000h	20480 bytes
Magic	0108h	PE32		Size of Headers	00000400h	
Linker Version	0C05h	5.12		Checksum	0000B499h	
Size of Code	00000400h			Subsystem	0002h	Win32 GUI
Size of Initialized Data	00000A00h			Dll Characteristics	0000h	
Size of Uninitialized Data	00000000h			Size of Stack Reserve	00100000h	
Address of Entry Point	00401000h			Size of Stack Commit	00001000h	
Base of Code	00001000h			Size of Heap Reserve	00100000h	
Base of Data	00002000h			Size of Heap Commit	00001000h	
Image Base	00400000h			Loader Flags	00000000h	Obsolete
				Number of Data Directories	00000010h	

[그림3-3. RegisterMe.exe의 PE Header 정보 1]

DATA DIRECTORIES				
Export Table	00900000	00050000	✓	Set to Zero
Directory Name	Virtual Address	Size		
Export Table	00900000h	00050000h		
Import Table	00402050h	0000003Ch		
Resource Table	00404000h	0000039Ch		
Exception Table				
Certificate Table				
Relocation Table				
Debug Data				
Architecture-specific data				
Machine Value (MIPS GP)				
TLS Table				
Load Configuration Table				
Bound Import Table				
Import Address Table	00402000h	00000050h		
Delay Import Descriptor				
COM+ Runtime Header				
(15) Reserved				

[그림3-4. RegisterMe.Oops.exe의 PE Header 정보 2]

DATA DIRECTORIES				
Export Table	00000000	00000000	✓	Set to Zero
Directory Name	Virtual Address	Size		
Export Table				
Import Table	00402050h	0000003Ch		
Resource Table	00404000h	0000039Ch		
Exception Table				
Certificate Table				
Relocation Table				
Debug Data				
Architecture-specific data				
Machine Value (MIPS GP)				
TLS Table				
Load Configuration Table				
Bound Import Table				
Import Address Table	00402000h	00000050h		
Delay Import Descriptor				
COM+ Runtime Header				
(15) Reserved				

[그림3-5. RegisterMe.exe의 PE Header 정보 2]

[그림3-2]와 [그림3-3] 그리고 [그림3-4]와 [그림3-5]를 비교해보면 값이 약간 다른 몇몇 Field Name과 Directory Name을 발견 할 수 있을 것입니다. 바로 Size of Code, Size of Initialized Data, Base of Code, Base of Data, Number of Data Directories, Export Table 입니다. 그러면 이제 변경된 PE Header data를 다시 변경해주는 [표3-1]의 Script에 대해서 알아보도록 하겠습니다.

Line 10 ~ 13

```
def main():
    imm = immlib.Debugger()
    module = imm.getModule(imm.getDebuggedName())
    pe = pefile.PE(module.getPath())
```

#### **imm = immlib.Debugger()**

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

#### **module = imm.getModule(imm.getDebuggedName())**

imm.getDebuggedName method로 현재 Debugging하고 있는 프로그램의 이름을 얻어오고, imm.getModule method로 그 프로그램의 정보를 얻어옵니다.

#### **pe = pefile.PE(module.getPath())**

얻어온 프로그램의 정보 중 getPath method를 이용하여 절대경로를 구하고, pefile.PE method를 사용하여 pe라는 변수에 절대경로에 해당하는 파일의 PE정보를 얻어옵니다.

Line 18 ~ 22

```
pe.OPTIONAL_HEADER.SizeOfCode = 0x400
pe.OPTIONAL_HEADER.SizeOfInitializedDate = 0xA00
pe.OPTIONAL_HEADER.BaseOfCode = 0x1000
pe.OPTIONAL_HEADER.BaseOfData = 0x2000
pe.OPTIONAL_HEADER.NumberOfRvaAndSizes = 0x10
```

#### **pe.OPTIONAL\_HEADER.SizeOfCode = 0x400**

pe라는 변수에 얻어온 PE정보 중에서 OPTIONAL\_HEADER의 SizeOfCode에 해당하는 값을 0x400으로 변경합니다.

#### **pe.OPTIONAL\_HEADER.SizeOfInitializedDate = 0xA00**

pe라는 변수에 얻어온 PE정보 중에서 OPTIONAL\_HEADER의 SizeOfInitializedDate에 해당하는 값을 0xA00으로 변경합니다.

#### **pe.OPTIONAL\_HEADER.BaseOfCode = 0x1000**

pe라는 변수에 얻어온 PE정보 중에서 OPTIONAL\_HEADER의 BaseOfCode에 해당하는 값을 0x1000으로 변경합니다.

#### **pe.OPTIONAL\_HEADER.BaseOfData = 0x2000**

pe라는 변수에 얻어온 PE정보 중에서 OPTIONAL\_HEADER의 BaseOfData에 해당하는 값을 0x2000으로 변경합니다.

#### **pe.OPTIONAL\_HEADER.NumberOfRvaAndSizes = 0x10**

pe라는 변수에 얻어온 PE정보 중에서 OPTIONAL\_HEADER의 NumberOfRvaAndSizes에 해당하는 값을 0x10으로 변경합니다.

Line 24 ~ 27

```
pe.OPTIONAL_HEADER.DATA_DIRECTORY[0].VirtualAddress = 0
pe.OPTIONAL_HEADER.DATA_DIRECTORY[0].Size = 0
pe.write('RegisterMe.Oops.1.exe')
```

#### **pe.OPTIONAL\_HEADER.DATA\_DIRECTORY[0].VirtualAddress = 0**

pe라는 변수에 얻어온 PE정보 중에서 OPTIONAL\_HEADER의 DATA\_DIRECTORY[0]에서 VirtualAddress에 해당하는 값을 0으로 변경합니다.

#### **pe.OPTIONAL\_HEADER.DATA\_DIRECTORY[0].Size = 0**

pe라는 변수에 얻어온 PE정보 중에서 OPTIONAL\_HEADER의 DATA\_DIRECTORY[0]에서 Size에 해당하는 값을 0으로 변경합니다.

#### **pe.write('RegisterMe.Oops.1.exe')**

변경된 내용들을 pe.write method를 이용하여 RegisterMe.Oops.1.exe라는 이름으로 Immunity Debugger가 설치된 곳에 저장합니다.

이 Script를 통해서 저장된 RegisterMe.Oops.1.exe를 Immunity Debugger를 사용하여 열게 되면 처음과는 다르게 잘 열리게 됩니다.

이러한 PE Header Modifications은 표준 값에서 벗어난 이상한 값들을 찾아내어 수정해주면 해결할 수 있습니다. 이러한 PE Header Modifications를 포함한 여러 가지 Anti-Debugging tricks를 회피하는 Plug-in이 있습니다. Olly Debugger용으로는 advancedolly.dll이 있지만, 아직 Immunity Debugger용으로 만들어진 것은 없는 것 같습니다.

## 0x04. 유용한 PyCommand

이번 장에서는 몇 가지의 유용한 PyCommand를 소개하려고 합니다. 그 첫 번째는 !scanpe입니다. scanpe.py는 Immunity Debugger가 설치된 디렉터리 밑 PyCommand디렉터리 안에 있습니다. PyCommand디렉터리에는 scanpe.py뿐만 아니라 다른 여러 PyCommand가 존재하고 있으므로 잘 살펴보시면 도움이 될만한 것들이 많을 거라고 생각합니다.

!scanpe는 Team PEiD의 Bob이라는 분이 만든 Script입니다. !scanpe가 하는 일은 UserDB.txt에 저장된 시그니처(Signatures)를 로딩해서 목표파일의 EP를 Scan후 로딩된 시그니처와 비교해서 그 결과를 알려주는 아주 유용한 Command입니다. PEiD툴과 유사하며 사용법은 다음과 같습니다.

1. [그림4-1]처럼 목표 파일을 Immunity Debugger로 엽니다.
2. Immunity Debugger창 아래 CommandLine에서 !scanpe를 입력하고 엔터를 칩니다.
3. [그림4-3]처럼 Alt+L을 사용하여 Log창으로 이동하여 그 결과를 봅니다.

[그림4-1. UPX로 압축된 Crackme파일]

[그림4-2. scanpe명령 CommandLine에 입력하기]

[그림4-3. Log창에 나타나는 결과]

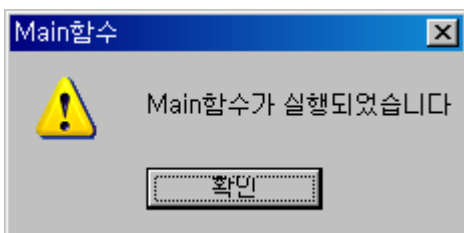
!scanpe는 공개된 시그니처 파일을 사용하기 때문에 확장이 용이합니다. UserDB.txt라는 파일은 <http://www.peid.info/BobSoft/UserDB.zip>에서 다운 받을 수 있으며, 로컬에는 Immunity Debugger가 설치된 디렉터리 밑 Data디렉터리 안에서 찾을 수 있습니다.



두 번째 소개드릴 PyCommand는 !patch입니다. patch.py는 앞서 설명했던 scanpe.py가 있는 곳에 함께 있습니다. !patch가 하는 일은 Anti-Debugging protection을 찾아서 protection을 해제시키는 일을 합니다. 먼저 예제프로그램을 분석하여 어떤 Anti-Debugging protection을 사용하는지 확인하겠습니다.

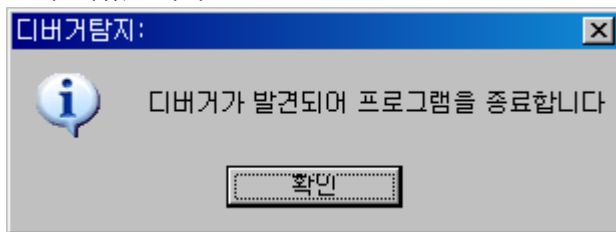
IsDebuggerPresent\_TLS.exe 3KB 응용 프로그램

[그림4-4. 예제프로그램]

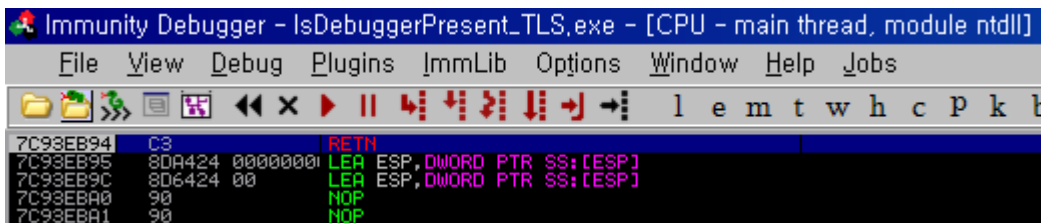


[그림4-5. 예제프로그램이 정상적으로 실행 되었을 때 뜨는 창]

정상적으로 예제프로그램이 실행이 되면 [그림4-5]처럼 Main함수가 실행되었다는 창이 뜹니다. 그리고 확인을 누르게 되면 프로그램은 종료됩니다. 그러면 이제 Immunity Debugger를 사용하여 예제프로그램을 열어 보도록 하겠습니다.



[그림4-6. 예제프로그램이 Debugger가 접근한걸 알았을 때 뜨는 창]



[그림4-7. 어색하기 짝이 없는 주소]

Immunity Debugger로 예제프로그램을 열어보았더니 도중에 [그림4-6]과 같이 “디버거가 발견되어 프로그램을 종료합니다”라는 창을 볼 수 있었습니다. 그리고 확인을 눌렀더니 Debugger가 [그림4-7]처럼 이상한 주소를 EP(Entry Point)로 잡고 보여주고 있습니다. 그래도 일단 예제프로그램이 디스어셈블리(Disassembly)된 코드를 확인하기 위해 메뉴 아이콘 중에서 'e'를 클릭하여 Executable modules창을 열어보겠습니다.

Base	Size	Entry	Name	File version	Path
00400000	00004000	00401000	IsDebugger		C:\project\IsDebuggerPresent_TLS.exe
5A700000	00000000	5A700000			
5A480000	00038000	5A481626			
5C820000	00097000	5C8232DA			
62340000	00009000	62342EAD			
73F80000	0006B000	73F8AEB6			
74660000	0004B000	746613A5			
75110000	0002E000	75129FCC			
762E0000	0001D000	762E12C0			
76970000	0013C000	769820C1			
770D0000	0008C000	770D1558			
77160000	00102000	771642B3			
77380000	007FA000	7739FA10			
77BC0000	00058000	77BCF2A1			
77CF0000	0008F000	77D00EB9			
77D80000	00091000	77D86284			
77E20000	00047000	77E2658A			
77E70000	00076000	77E751D3			
77F50000	000A8000	77F578D4			
78130000	0009B000	78132328			
7C420000	00087000	7C450DCE			
7C800000	0012E000	7C80B436			
7C930000	0009C000	7C943156			

[그림4-8. Executable modules 창]

메뉴 아이콘 중에서 'e'를 클릭하여 Executable modules창을 열게 되면 [그림4-8]처럼 창이 열립니다. 빨간색 테두리가 쳐진 예제프로그램이 있는 라인을 더블 클릭하게 되면 CPU창에 해당 코드가 보이게 됩니다.

```

00401000 6A 30          PUSH 30
00401002 68 36304000   PUSH IsDebugger.00403036
00401007 68 40304000   PUSH IsDebugger.00403040
0040100C 6A 00          PUSH 0
0040100E E8 3D000000   CALL <JMP.&user32.MessageBoxA>
00401013 6A 00          PUSH 0
00401015 E8 3C000000   CALL <JMP.&kernel32.ExitProcess>
0040101A C3            RETN
0040101B 803D 60304000 CMP BYTE PTR DS:[403060],1
00401022 74 2B        JE SHORT IsDebugger.0040104F
00401024 C605 60304000 MOV BYTE PTR DS:[403060],1
0040102B E8 2C000000   CALL <JMP.&kernel32.IsDebuggerPresent>
00401030 83F8 01      CMP EAX,1
00401033 75 1A        JNZ SHORT IsDebugger.0040104F
00401035 6A 40          PUSH 40
00401037 68 00304000   PUSH IsDebugger.00403000
0040103C 68 0D304000   PUSH IsDebugger.0040300D
00401041 6A 00          PUSH 0
00401043 E8 80000000   CALL <JMP.&user32.MessageBoxA>
00401048 6A 00          PUSH 0
0040104A E8 07000000   CALL <JMP.&kernel32.ExitProcess>
0040104F C3            RETN
00401050 FF25 0C204000 JMP DWORD PTR DS:[&user32.MessageBoxA]
00401056 FF25 04204000 JMP DWORD PTR DS:[&kernel32.ExitProcess]
0040105C FF25 00204000 JMP DWORD PTR DS:[&kernel32.IsDebuggerPresent]
  
```

[그림4-9. 더블클릭 후 찾아간 예제프로그램의 OEP(Original Entry Point)]

[그림4-9]의 코드 중간 쪼에서 IsDebuggerPresent라는 API를 볼 수 있습니다.

```

BOOL IsDebuggerPresent(void);
//IsDebuggerPresent API는 parameters가 없음
//현재 Process가 Debugger에서 실행 중이라면 zero가 아닌 값을 리턴하고, Debugger에서 실행 중이
아니라면 zero값을 리턴함
  
```

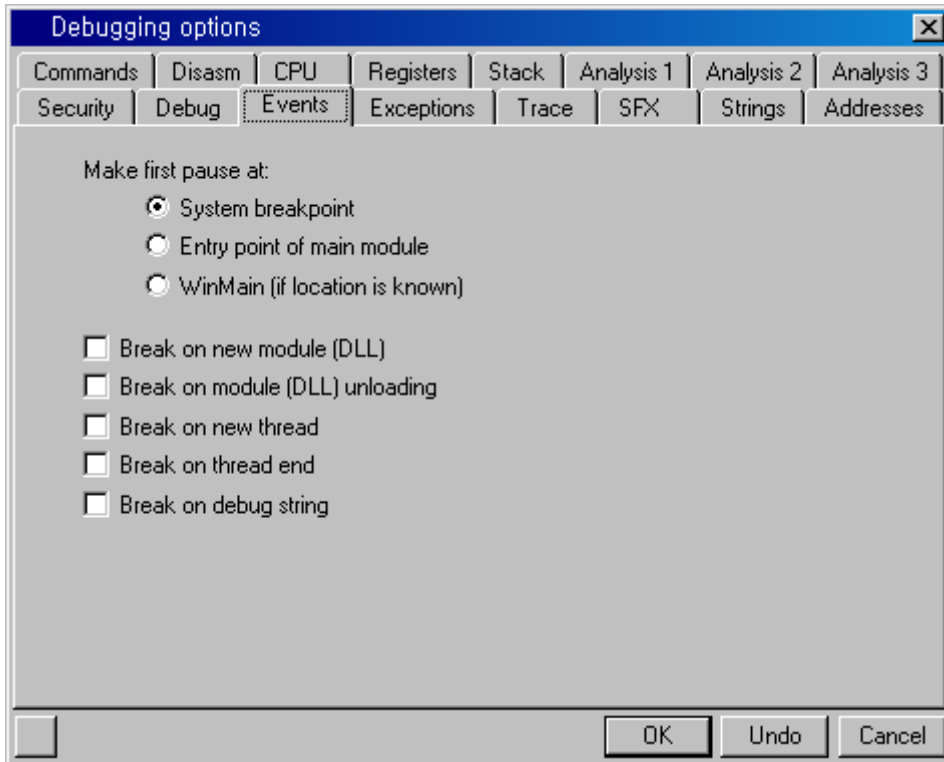
[표4-1. IsDebuggerPresent API]

```

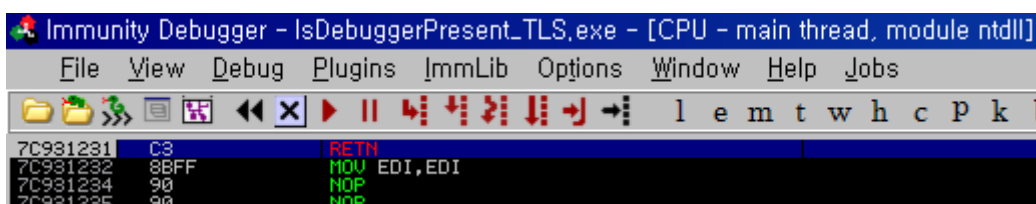
0040102B      CALL <JMP.&kernel32.IsDebuggerPresent>
//IsDebuggerPresent API를 호출 합니다.
//만약 Debugger에서 현재 Process가 진행 중이라면 0이 아닌 값을 리턴 합니다.
00401030      CMP EAX,1
//따라서 EAX Register에 1이 들어가게 되고 1과 비교하게 됩니다.
00401033      JNZ SHORT IsDebugger.0040104F
//00401030에서 비교된 결과로 Zeroflag가 0에서 1로 변경됩니다. 그래서 분기하지 않습니다.
  
```

00401033에서 분기하지 않고 진행하게 되면 [그림4-6]처럼 디버거가 탐지 되었다는 창을 보게 됩니다. 이것을 해결 할 수 있는 방법은 여러 가지가 있겠지만 PyCommand인 !patch를 사용하여 해결하도록 하겠습니다.

우선 Immunity Debugger를 실행시켜 메뉴에서 Option의 Debugger Options(Alt+O)에 들어갑니다. 많은 탭이 있는 창이 하나 열리는데 [그림4-10]처럼 설정을 변경합니다. System breakpoint로 변경을 하는 이유는 IsDebuggerPresent때문에 기존의 옵션으로는 예제프로그램의 EP에서 멈출 수 없기 때문입니다.



[그림4-10. Debugger Options]



[그림4-11. System Breakpoint]

[그림4-10]처럼 설정을 변경하고 예제프로그램을 열게 되면 [그림4-11]같이 System Breakpoint 라는 곳에서 멈추게 됩니다. 그럼 이제 CommandLine에서 !patch명령을 사용하도록 하겠습니다.



[그림4-12. patch명령 CommandLine에 입력 전후]

!patch명령을 실행 한 후에 [그림4-12]의 IsDebuggerPresent patched 메시지를 확인한 후 F9를 눌러보면 예제프로그램의 OEP에 도달 할 수 있게 됩니다.

그러면 이제 patch.py를 분석해보도록 하겠습니다. Block Comment는 생략하겠습니다.

```
#!/usr/bin/env python
"""
Immunity Debugger Patcher

(c) Immunity, Inc. 2004-2007

U{Immunity Inc.<http://www.immunityinc.com>}
"""
__VERSION__ = '1.1'
NOTES="""
anti-antidebugging is here
DONE: IsDebuggerPresent
TODO:
* EnumProcesses
* CreateToolhelp32Snapshot, Process32First, Process32Next,
* UnhandledExceptionFilter - ZwQueryInformationProcess
* ProcessHeapFlag & NTGlobalFlag
"""

import immllib
from immllib import BpHook
import getopt

DESC="Patches anti-debugging protection , [-t TYPE_OF_PROTECTION]"

def usage(imm):
    imm.Log("!patch -t TYPE",focus=1)

def main(args):
    types={"isdebuggerpresent": 0}
    imm = immllib.Debugger()

    if not args:
        return "give patch type..."

    try:
        opts, argo = getopt.getopt(args, "t:s")
    except getopt.GetoptError:
        usage(imm)
        return "Bad patch argument %s" % args[0]

    type = None

    for o,a in opts:
        if o == '-t':
            low = a.lower()
            if types.has_key( low ):
                type = types[ low ]
            else:
                return "Invalid type: %s" % a

    # IsDebuggerPresent
    if type == 0:
        imm.Log( "Patching IsDebuggerPresent..." )
        ispresent = imm.getAddress( "kernel32.IsDebuggerPresent" )
        imm.writeMemory( ispresent, imm.Assemble( "xor eax, eax\n ret" ) )

        return "IsDebuggerPresent patched"

    else:
        usage(imm)
        return "Bad patch argument"
```

Line 24 ~ 28

```
import immlib
from immlib import BpHook
import getopt

DESC="Patches anti-debugging protection , [-t TYPE_OF_PROTECTION]"
```

### **import immlib, import getopt**

import를 이용해 immlib, getopt module을 사용하게 합니다. immlib는 Debugging용 Class를 사용하기 위한 것이고, getopt는 main함수의 parameter인 args에 받는 CommadLine의 옵션을 분리하기 위해서 사용합니다.

### **from immlib import BpHook**

**from** module **import** name1,[name2,[... name N]]을 사용하여 immlib모듈내의 BpHook을 현재의 네임 스페이스로 불러들입니다.

### **DESC="Patches anti-debugging protection , [-t TYPE\_OF\_PROTECTION]"**

DESC라는 변수에 "Patches ... [-t TYPE\_OF\_PROTECTION]"이라는 문자열을 넣습니다.

Line 30 ~ 31

```
def usage(imm):
    imm.Log("!patch -t TYPE",focus=1)
```

usage라는 이름을 가지고 imm이라는 parameter를 가지는 함수를 선언합니다. 이 함수가 호출되면 Immunity Debugger Log창에 "!patch -t TYPE"라는 로그를 남기게 됩니다.

Line 33 ~ 45

```
def main(args):
    types={"isdebuggerpresent": 0}
    imm = immlib.Debugger()

    if not args:
        return "give patch type..."

    try:
        opts, argo = getopt.getopt(args, "t:s")
    except getopt.GetoptError:
        usage(imm)
        return "Bad patch argument %s" % args[0]
```

### **def main(args):**

args라는 parameter를 가지고 있는 main함수 입니다.

### **types={"isdebuggerpresent": 0}**

types라는 변수에 사전을 만드는데 isdebuggerpresent라는 키에 0을 할당합니다. 사전이란 Python의 유일한 Mapping형으로서 '키:값' 형태로 표현됩니다.

### **imm = immlib.Debugger()**

debugging용 method를 사용하기 위해 immlib의 Debugger Class를 imm이라는 변수로 선언하였습니다.

if not args:

**return "give patch type..."**

main함수가 parameter값 없이 실행된다면 "give patch type..."을 return하게 됩니다.

try:

**opts, argo = getopt.getopt(args, "t:s")**

try-except 구문의 try부분입니다. try-except 구문은 예외를 정의하여 예외처리를 할 수 있는 메커니즘을 제공합니다. getopt 함수는 2개의 인자를 가지는데 첫 번째 인자인 args는 인수리스트를 뜻하고 두 번째 인자(argument)인 "t:s"는 옵션문자를 뜻합니다. 옵션문자 't'뒤에 ':'(colon)이 있으므로 't'옵션은 인자를 갖는 옵션이 됩니다. 그리고 옵션문자 's'는 ':'(colon)이 없으므로 단독 옵션이 됩니다. 즉, 't'옵션문자는 인자를 가지는데 's'옵션문자는 인자를 가지지 못합니다.

```
>>> import getopt
>>> args = '-t isdebuggerpresent -s whoami'.split()
>>> opts, argo = getopt.getopt(args, 't:s')
>>> opts
[('-t', 'isdebuggerpresent'), ('-s', '')]
>>> argo
['whoami']
>>> _
```

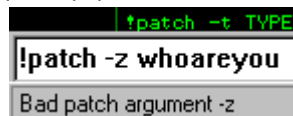
[그림4-13. 옵션문자 처리]

except getopt.GetoptError:

**usage(imm)**

**return "Bad patch argument %s" % args[0]**

try-except 구문의 except부분입니다. try부분에서 정의한 것에서 벗어나는 것에 대해서는 모두 예외처리를 하게 됩니다. 우선 usage함수를 호출하여 [그림4-14]처럼 Immunity Debugger Log창에 "!patch -t TYPE"라는 로그를 남기며, 정의되지 않은 명령행 옵션에 대해서는 "Bad patch argument ..."라는 문자열을 리턴 하게 됩니다.



```
!patch -t TYPE
!patch -z whoareyou
Bad patch argument -z
```

[그림4-14. except]

Line 47 ~ 68

```
type = None

for o,a in opts:
    if o == '-t':
        low = a.lower()
        if types.has_key( low ):
            type = types[ low ]
        else:
            return "Invalid type: %s" % a

# IsDebuggerPresent
if type == 0:
    imm.Log( "Patching IsDebuggerPresent..." )
    ispresent = imm.getAddress( "kernel32.IsDebuggerPresent" )
    imm.writeMemory( ispresent, imm.Assemble( "xor eax, eax\n ret" ) )
```

```

return "IsDebuggerPresent patched"

else:
    usage (imm)
    return "Bad patch argument"

```

### type = None

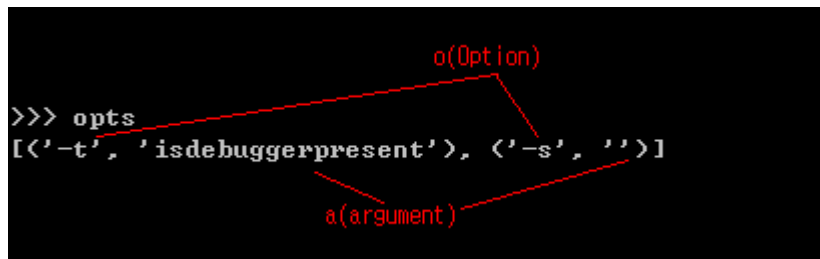
type이라는 변수를 선언하는데 None이라는 특별한 타입으로 인해 type변수는 어떤 속성도 값도 가지지 않습니다.

### for o,a in opts:

```
if o == '-t':
```

```
    low = a.lower()
```

for구문입니다. [그림4-15]를 통해서 설명하겠습니다. opts는 [('-t', 'isdebuggerpresent'), ('-s', '')]를 값으로 가지고 있습니다. for문에 있는 'o'와 'a'는 'Option'과 'Argument'의 줄임 말로 보시면 됩니다.



[그림4-15. option, argument]

만약에 'o'가 '-t'와 같다면 '-t'의 인자(argument)인 'isdebuggerpresent'를 low라는 변수에 넣는데 lower함수를 써서 '-t'의 인자를 소문자로 변경시킨 후 넣게 됩니다.

### if types.has\_key( low ):

```
    type = types[ low ]
```

else:

```
    return "Invalid type: %s" % a
```

그리고 has\_key함수를 사용하여 types의 키와 low변수에 저장된 값을 비교하여 참, 거짓을 판단합니다. 비교한 결과가 참이라면 type변수에 types의 키가 할당 받은 값인 0을 넣게 됩니다. 그렇지 않고 거짓이라면 "Invalid type: %s" % a 를 리턴 하여 [그림4-16]과 같은 결과를 나타냅니다.



[그림4-16. Invalid type]

### if type == 0:

```
imm.Log( "Patching IsDebuggerPresent..." )
```

```
ispresent = imm.getAddress( "kernel32.IsDebuggerPresent" )
```

```
imm.writeMemory( ispresent, imm.Assemble( "xor eax, eax\n ret" ) )
```

```
return "IsDebuggerPresent patched"
```

patch.py에서 가장 중요한 부분입니다. CommandLine에서 올바른 명령을 넣고 모든 코드가 정상적으로 진행이 되면 마지막으로 위의 if문이 실행이 됩니다. 우선 Immunity Debugger Log창에 "Patching IsDebuggerPresent..."를 남기고, imm.getAddress method로 ispresent라는 변수에 kernel32.IsDebuggerPresent의 주소를 얻어 옵니다.

```

Immunity Debugger Python Shell
*** Immunity Debugger Python Shell v0.1 ***
Immlib instanced as 'imm' PyObject
READY.
>>>import immlib
>>>imm = immlib.Debugger()
>>>ispresent = imm.getAddress("kernel32.IsDebuggerPresent")
>>>ispresent
2088840707
>>>hex(ispresent)
'0x7c812e03'
>>>

```

[그림4-17. getAddress("kernel32.IsDebuggerPresent")]

그리고 imm.writeMemory method로 ispresent변수가 가지고 있는 주소(0x7C812E03)에 imm.Assemble method로 "xor eax, eax\n ret"라는 Assemble code를 씁니다. [그림4-18]과 [그림4-19]를 비교해 보면 해당주소의 Assemble code가 어떻게 변했는지 알 수 있습니다.

이렇게 imm.writeMemory method까지 정상적으로 끝나게 되면 "IsDebuggerPresent Patched"라는 성공메시지를 리턴 합니다.

```

CPU - main thread, module kernel32
7C812E03 64:A1 18000000 MOV EAX, DWORD PTR FS:[18]
7C812E09 8B40 30 MOV EAX, DWORD PTR DS:[EAX+30]
7C812E0C 0FB640 02 MOVZX EAX, BYTE PTR DS:[EAX+2]
7C812E10 C3 RETN
7C812E11 90 NOP
7C812E12 90 NOP
7C812E13 90 NOP
7C812E14 90 NOP
7C812E15 90 NOP
7C812E16 8BFF MOV EDI, EDI

```

[그림4-18. writeMemory 전 0x7C812E03]

```

CPU - main thread, module kernel32
7C812E03 33C0 XOR EAX, EAX
7C812E05 C3 RETN
7C812E06 0000 ADD BYTE PTR DS:[EAX], AL
7C812E08 008B 40300FB6 ADD BYTE PTR DS:[EBX+B60F3040], CL
7C812E0E 40 INC EAX
7C812E0F 02C3 ADD AL, BL
7C812E11 90 NOP
7C812E12 90 NOP
7C812E13 90 NOP
7C812E14 90 NOP
7C812E15 90 NOP
7C812E16 8BFF MOV EDI, EDI

```

[그림4-19. writeMemory 후 0x7C812E03]

IsDebuggerPresent API에 대한 패치가 끝났습니다. 수정된 IsDebuggerPresent API가 호출되면 EAX Register는 자기자신을 XOR시켜 0으로 초기화시킵니다. 그러면 [그림4-9]의 0x0040102B ~ 0x00401033에서 EAX는 0이므로 IsDebuggerPresent API는 무력화 되고 프로그램은 정상적으로 main함수를 호출 하게 됩니다.



## 0x05. 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
- **열혈강의 Python** – 이강성 저  
Python 프로그래밍 서적
- **해킹, 파괴의 광학(개정판)** – 김성우 저  
윈도우 프로그래밍 서적
- **pefile module** – <http://code.google.com/p/pefile>  
pefile module 페이지
- **Anti Reverse Engineering Uncovered** – <http://www.codebreakers-journal.com/>  
Code Breaker Journal
- **Lena151 tutorial 03 Script** - <http://forum.immunityinc.com/index.php?topic=141.0>  
본 문서에서 사용하는 script
- **reverseMe.exe** - <http://www.tuts4you.com/download.php?view.124>  
본 문서에서 사용하는 예제프로그램
- **IsDebuggerPresent\_TLS.exe** - <http://zesrever.xstone.org/9>  
본 문서에서 사용하는 예제프로그램 및 TLS CallBack