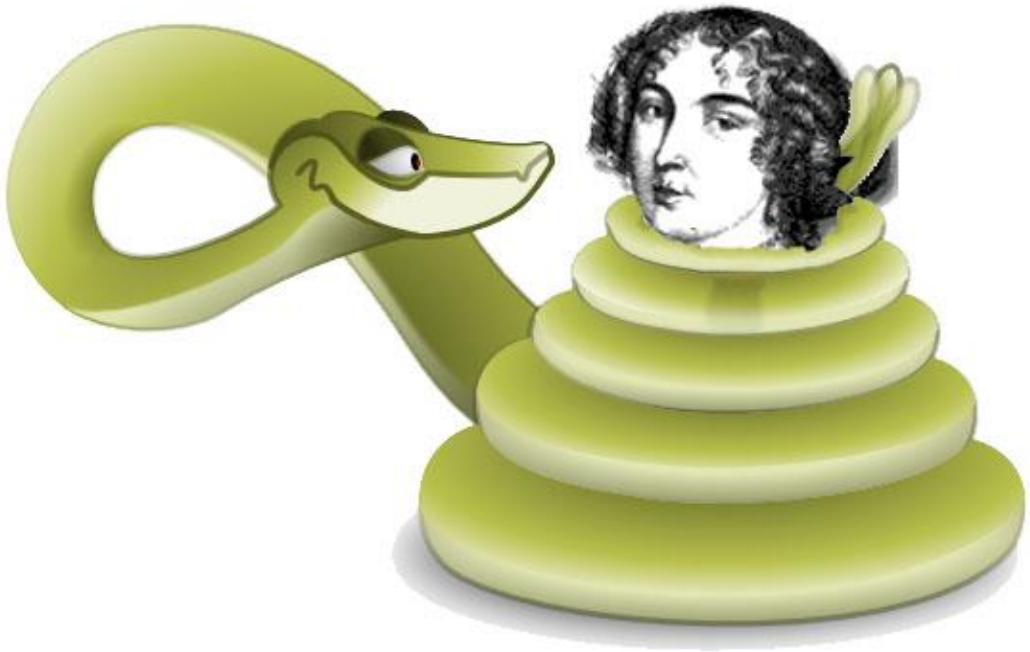
IDAPython

Python plugin for Interactive Disassembler



suspected.tistory.com

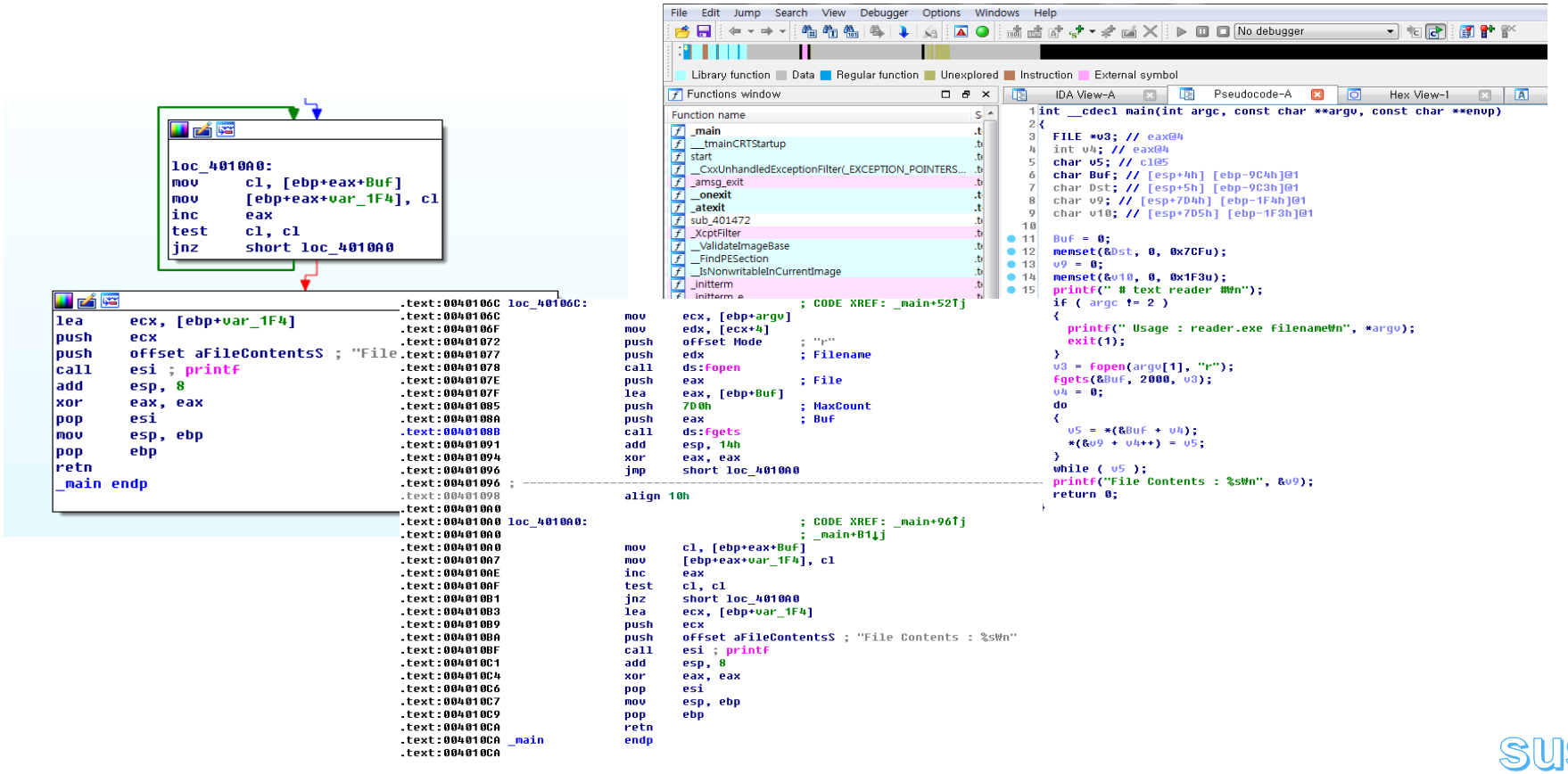
목차

- IDAPython 소개 및 설치
- IDAPython API
- IDAPython 활용

IDAPython 소개 및 설치

IDA Pro

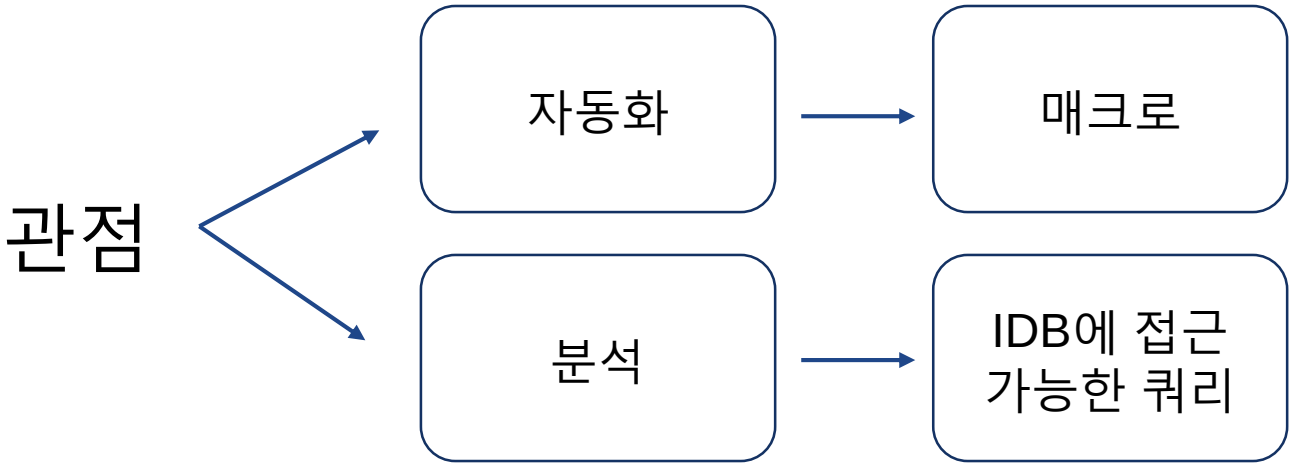
IDA Pro : Hex-rays 사의 Disassembler & Debugger



IDA 스크립트 소개

IDC : C언어와 매우 유사한 IDA의 내장 스크립트

IDAPython : IDC 기반으로 만들어진 Python 스크립트



* IDB(IDA Database File)

IDAPython 설치 방법

실습 버전 :	IDA Pro Version : 6.9.160222 IDAPython Version : 1.7.0 Python Version : 2.7.6 OS : Windows 64bit

1. 파이썬 2.7 32bit 설치(https://www.python.org/downloads/)
2. IDA Pro Demo Version 설치(http://out7.hex-rays.com/demo/request)
3. IDAPython 설치(https://github.com/idapython/src)
4. IDAPython 파일 압축 해제
5. IDAPython 내부 python 디렉터리를 IDA가 설치된 폴더로 복사
6. plugins 폴더 안에 있는 파일들을 IDA Pro의 plugins 폴더로 복사
7. python.cfg 파일을 IDA Pro의 cfg 폴더 내로 복사

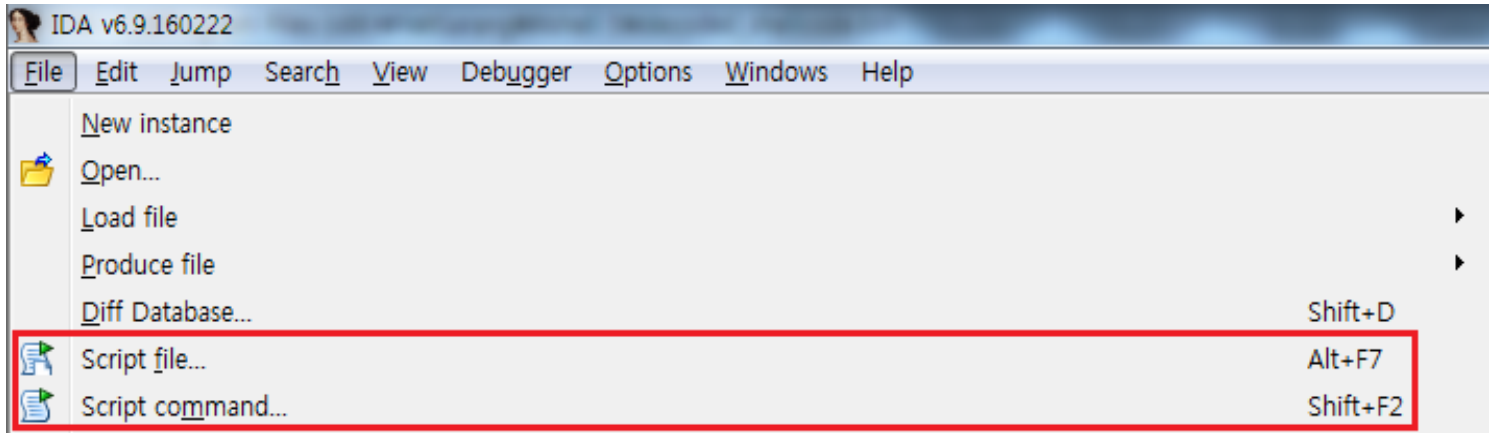
IDA pro 라이센스가 있을 경우에는 자동으로 설치됨으로 위 과정 패스

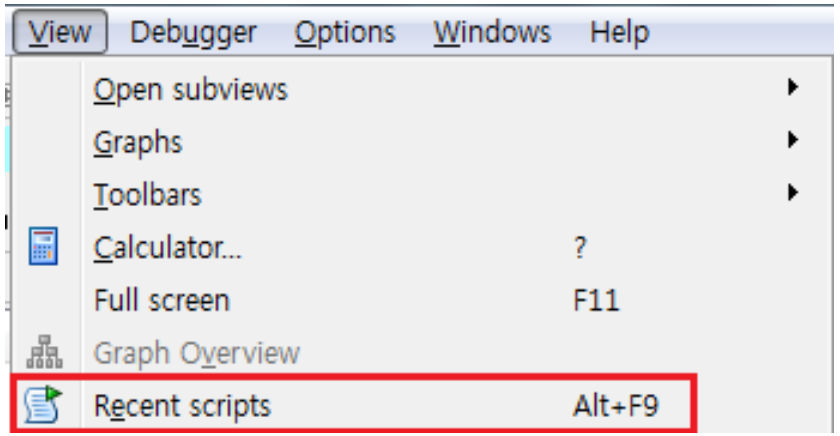
IDAPython 실행

Alt + F7 : 스크립트파일 가져와 실행

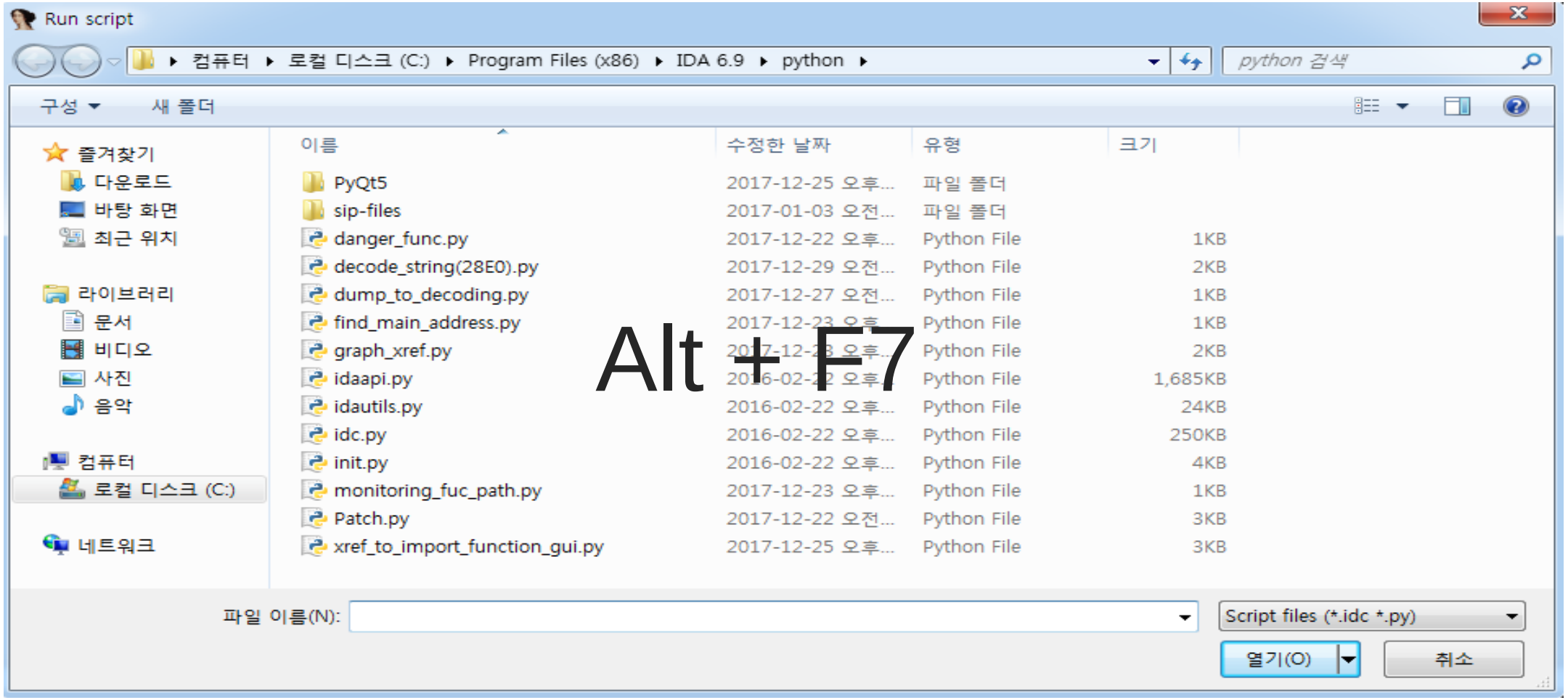
Shift + F2 : 스크립트 명령창 띄움(테스트 코드 작성 시 유용!)

Alt + F9 : 최근 실행한 스크립트 확인



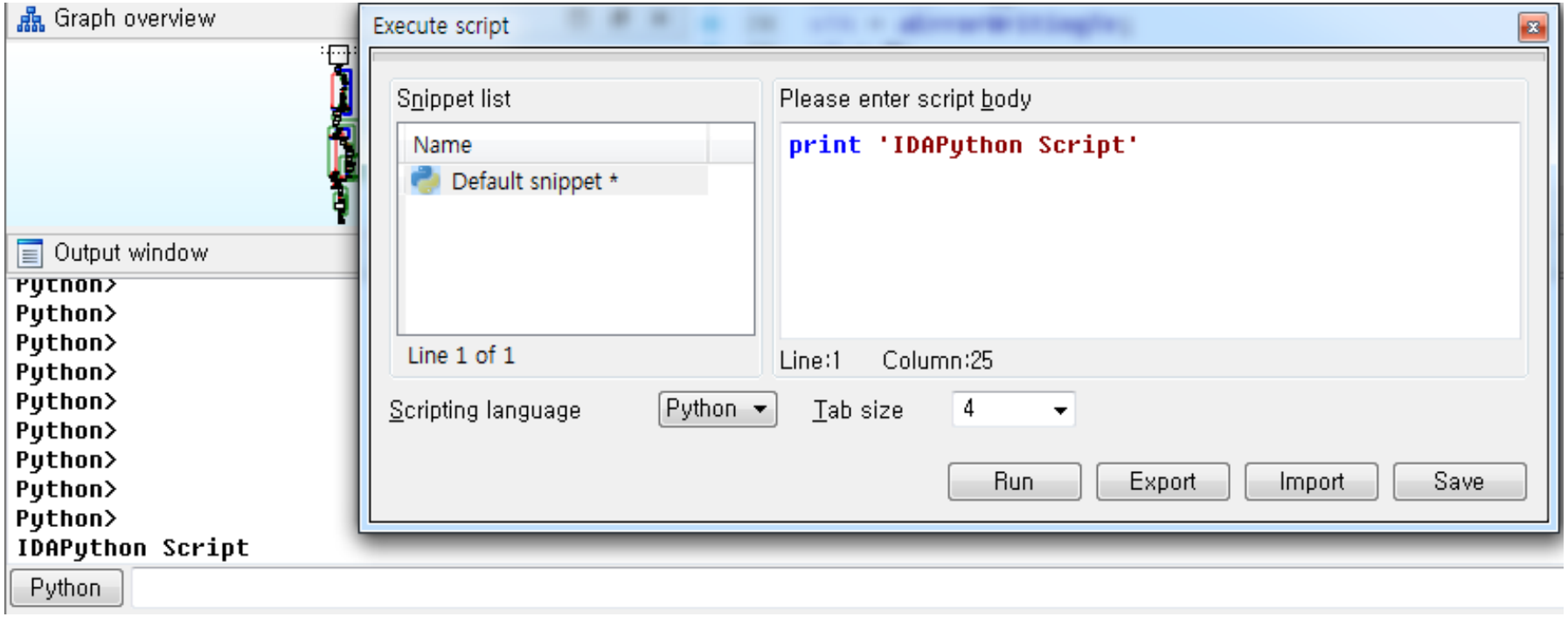


IDAPython 실행



IDAPython 실행

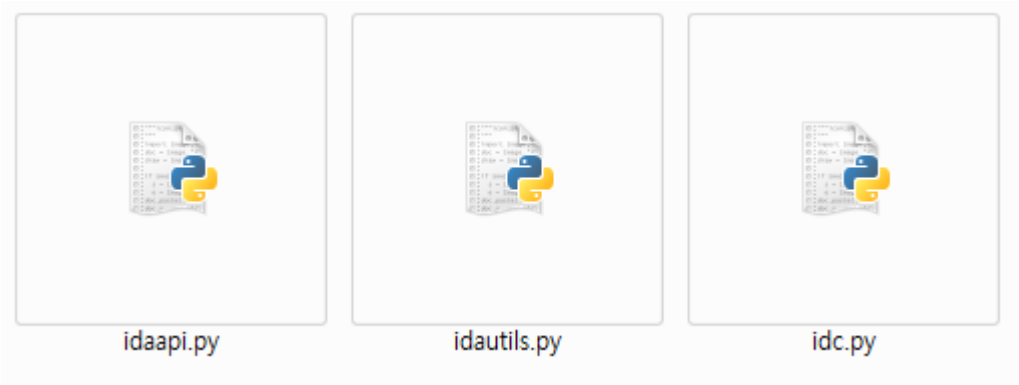
Shift + F2



IDAPython API

IDAPython API

경로 : %IDA_DIR%\\Python



코드 상단에 작성 추천

```
from idc import *
from idautils import *
from idaapi import *
```

ScreenEA, MinEA, MaxEA

idc.ScreenEA() : 현재 위치의 주소를 리턴

idc.here()

```
Python>hex(ScreenEA()), hex(here())
('0x4011f4L', '0x4011f4L')
```

Python

idc.MinEA(address) : 최소 주소를 리턴

idc.MaxEA(address) : 최대 주소를 리턴

```
Python>hex(MinEA())
0x401000L
Python>hex(MaxEA())
0x40e000L
```

Get Segment Values

세그먼트(Segment) : 프로그램에 정의된 특정 영역으로 코드, 데이터, 스택이 존재

- idc.SegName(address) : address에 대한 세그먼트 이름을 리턴
- idc.SegStart(address) : address에 대한 세그먼트 시작 주소를 리턴
- idc.SegEnd(address) : address에 대한 세그먼트 마지막 주소를 리턴

```
for seg in idautils.Segments():
 print idc.SegName(seg), idc.SegStart(seg), idc.SegEnd(seg)
```

```
Python>
.text 4198400 4214784
.idata 4214784 4214948
.rdata 4214948 4218880
.data 4218880 4222976
```

Get Function Name

idautils.Functions(start_addr, end_addr) : 함수 목록을 리스트 형태로 리턴(범위 지정 가능)

idc.GetFuncName(function_address) : 함수 주소의 이름을 리턴

```
for func in idautils.Functions():
 print hex(func), idc.GetFunctionName(func)
```

```
0x401000L sub_401000
0x401047L sub_401047
0x401191L sub_401191
0x4011d6L _WinMain@16
0x4013cdL sub_4013CD
0x4014f2L sub_4014F2
0x4017d2L sub_4017D2
0x401802L sub_401802
0x401824L sub_401824
0x4018a5L sub_4018A5
```

HotKey

- HotKey를 등록해 첫 번째 인자로 지정한 단축키로 원하는 함수 실행

```
def AddHotkey(hotkey, idcfunc):
    """
    Add hotkey for IDC function

    @param hotkey: hotkey name ('a', "Alt-A", etc)
    @param idcfunc: IDC function name

    @return: None
    """
    return idaapi.add_idc_hotkey(hotkey, idcfunc)
```

```
def first():
 print 'Alt-Z!!'

idaapi.add_hotkey("Alt-Z", first())
```

IDAPython – Main 함수 찾기

- 함수 이름을 모두 불러와 main, Main, Start, start 값이 포함되어 있으면 출력하는 스크립트

```
from idautils import *
from idaapi import *
from idc import *

ea = BeginEA() # 코드의 맨 앞부분(보통 0x00000000)

# Segment의 시작부터 끝까지 함수의 주소를 가져옴
for funcAddr in Functions(SegStart(ea), SegEnd(ea)):
    funcName = GetFunctionName(funcAddr) # 함수의 이름을 가져옴
    # 함수의 이름이 'main', 'Main', 'Start', 'start' 이면 해당 주소를 출력
    if funcName.find("main") > 0 or funcName.find("Main") > 0 or \
       funcName.find("Start") > 0 or funcName.find("start"):
        print "Function %s is at 0x%x" % (funcName, funcAddr)
```

Function name	Segment	Start
sub_401000	.text	00401000
sub_401047	.text	00401047
sub_401191	.text	00401191
WinMain(x,x,x,x)	**.text**	**004011D6**
sub_4013CD	.text	004013CD
sub_4014F2	.text	004014F2

Output window

```
Python>
Python>
Python>
Function _WinMain@16 is at 0x4011d6
Caching 'Imports'... ok
```

GetFunctionAttr

idc.GetFunctionAttr(ea, attr) : address의 attr(함수의 시작 or 끝 주소) 리턴
- attr : FUNCATTR_START(함수의 시작), FUNCATTR_END(함수의 끝)

```
def GetFunctionAttr(ea, attr):
    """
    Get a function attribute

    @param ea: any address belonging to the function
    @param attr: one of FUNCATTR_... constants

    @return: BADADDR - error otherwise returns the attribute value
    """
    func = idaapi.get_func(ea)

    return _IDC_GetAttr(func, _FUNCATTRMAP, attr) if func else BADADDR
```

```
Python>idc.GetFunctionAttr(ea, FUNCATTR_START)
4198870
Python>idc.GetFunctionAttr(ea, FUNCATTR_END)
4199373
```

GetDisasm

idc.GetDisasm(ea) : addres의 디스어셈블리를 리턴

```
def GetDisasm(ea):
    """
    Get disassembly line

    @param ea: linear address of instruction

    @return: "" - could not decode instruction at the specified location

    @note: this function may not return exactly the same mnemonics
           as you see on the screen.
    """
    return GetDisasmEx(ea, 0)
```

```
Python>idc.GetDisasm(here())
push    ebp
```

```
def GetDisasmEx(ea, flags):
    """
    Get disassembly line

    @param ea: linear address of instruction

    @param flags: combination of the GENDSM_ flags, or 0

    @return: "" - could not decode instruction at the specified location

    @note: this function may not return exactly the same mnemonics
           as you see on the screen.
    """
    text = idaapi.generate_disasm_line(ea, flags)
    if text:
        return idaapi.tag_remove(text)
    else:
        return ""
```

NextHead, PrevHead

idc.NextHead(address) : 다음 명령 주소(instruction address)를 리턴

```
def NextHead(ea, maxea=BADADDR):
    """
    Get next defined item (instruction or data) in the program

    @param ea: linear address to start search from
    @param maxea: the search will stop at the address
        maxea is not included in the search range

    @return: BADADDR - no (more) defined items
    """
    return idaapi.next_head(ea, maxea)
```

```
Python>here()
4198870
Python>idc.NextHead(here())
4198871
Python>idc.PrevHead(here())
4198867
```

idc.PrevHead(address) : 이전 명령 주소(instruction address)를 리턴

```
def PrevHead(ea, minea=0):
    """
    Get previous defined item (instruction or data) in the program

    @param ea: linear address to start search from
    @param minea: the search will stop at the address
            minea is included in the search range

    @return: BADADDR - no (more) defined items
    """
    return idaapi.prev_head(ea, minea)
```

IDAPython - Sample Script(1)

- 특정 함수의 시작부터 끝까지 디스어셈블 코드를 출력하는 스크립트

```
ea = here()
start = idc.GetFunctionAttr(ea, FUNCATTR_START)
end = idc.GetFunctionAttr(ea, FUNCATTR_END)
cur_addr = start
while cur_addr <= end:
 print hex(cur_addr), idc.GetDisasm(cur_addr)
 cur_addr = idc.NextHead(cur_addr, end)
```

```
0x4011d6L push    ebp
0x4011d7L mov     ebp, esp
0x4011d9L sub     esp, 1Ch
0x4011dcL mov     [ebp+var_18], 0
0x4011e3L mov     [ebp+var_4], 0
0x4011eaL call    sub_401047
0x4011efL mov     eax, dword_4067A4
0x4011f4L mov     ecx, [eax+90h]
0x4011faL push    ecx
0x4011fbL call    sub_401D7B
0x401200L add     esp, 4
```

GetFunctionFlags

idc.GetFunctionFlags(address) : address의 플래그 값을 리턴

```
def GetFunctionFlags(ea):
    """
    Retrieve function flags

    @param ea: any address belonging to the function

    @return: -1 - function doesn't exist otherwise returns the flags
    """
    func = idaapi.get_func(ea)

    if not func:
        return -1
    else:
        return func.flags
```

```
for func in idautils.Functions():
 flags = idc.GetFunctionFlags(func)
 if flags & FUNC_LIB:
  print hex(func), "FUNC_LIB", GetFunctionName(func)
```

```
0x401a96L FUNC_LIB __get_startup_argv_mode
0x401bdeL FUNC_LIB unknown_libname_1
0x401be8L FUNC_LIB unknown_libname_2
0x401f70L FUNC_LIB _memset
0x401fd0L FUNC_LIB _memcpy
0x402305L FUNC_LIB start
0x4023fbL FUNC_LIB __amsg_exit
0x402420L FUNC_LIB _fast_error_exit
0x402444L FUNC_LIB __cinit
0x402471L FUNC_LIB _exit
0x402482L FUNC_LIB __exit
```

GetFunctionFlags

Flags

- FUNC_LIB : 라이브러리 코드를 찾는 데 사용
- FUNC_STATIC : 정적 함수로 컴파일된 함수를 식별하는데 사용
- FUNC_FRAME : 프레임 포인터 push ebp를 사용하는 함수를 식별하는데 사용
- FUNC_BOTTOMBP : FUNC_FRAME과 마찬가지로 프레임 포인터 pop ebp를 추적하는데 사용
- FUNC_THUNK : 다른 함수로 점프하는 간단한 함수인 THUNK를 식별하는데 사용

FuncItems

- FuncItems(start) : 지정한 함수 주소에서 모든 주소를 리턴

```
def FuncItems(start):
    """
    Get a list of function items

    @param start: address of the function

    @return: ea of each item in the function
    """
    func = idaapi.get_func(start)
    if not func:
        return
    fii = idaapi.func_item_iterator_t()
    ok = fii.set(func)
    while ok:
        yield fii.current()
        ok = fii.next_code()
```

FuncItems의 리턴 값이 iterator 형태이기 때문에 list로 변환하면 쉽게 확인 가능

```
Python>FuncItems(here())
<generator object FuncItems at 0x058DF0D0>
```

```
Python>list(FuncItems(here()))
[4198870L, 4198871L, 4198873L, 4198876L, 4198883L, 4198890L
4198965L, 4198971L, 4198977L, 4198980L, 4198981L, 4198986L,
4199046L, 4199051L, 4199058L, 4199064L, 4199070L, 4199071L,
4199139L, 4199145L, 4199151L, 4199154L, 4199155L, 4199161L,
4199242L, 4199247L, 4199257L, 4199262L, 4199268L, 4199269L,
4199330L, 4199333L, 4199334L, 4199337L, 4199338L, 4199343L,
```

GetMnem

- GetMnem(address) : address의 레지스터를 리턴

```
Python>GetDisasm(here())
call    dword ptr [edx+1A0h]
Python>GetMnem(here())
call
```

```
def GetMnem(ea):
    """
    Get instruction mnemonics

    @param ea: linear address of instruction

    @return: "" - no instruction at the specified location

    @note: this function may not return exactly the same mnemonics
    as you see on the screen.
    """
    res = idaapi.ua_mnem(ea)

    if not res:
        return ""
    else:
        return res
```

GetOpnd

- GetOpnd(ea, n) : addres의 n번째 오퍼랜드 값을 리턴

```
Python>GetDisasm(here())
mov     [ebp+var_10], eax
Python>GetOpnd(here(), 0)
[ebp+var_10]
Python>GetOpnd(here(), 1)
eax
```

```
def GetOpnd(ea, n):
    """
    Get operand of an instruction

    @param ea: linear address of instruction
    @param n: number of operand:
        0 - the first operand
        1 - the second operand

    @return: the current text representation of operand or ""
    """

    if not isCode(idaapi.get_flags_novalue(ea)):
        return ""

    res = idaapi.ua_outop2(ea, n)

    if not res:
        return ""
    else:
        return idaapi.tag_remove(res)
```

- GetOpnd 함수는 내부적으로 ua_outop2 함수를 사용 후 tag_remove 함수로 문자열만 추출

```
Python>res = ua_outop2(here(), 0)
Python>res
┌)┌      [┐        ┌ebp┐ ┌  +┐        ┌+var_10┐ ├┌        ]┐        ┐)
Python>tag_remove(res)
[ebp+var_10]
```

- ua_outop2(ea, n) : address의 저수준 문자를 가져옴
- tag_remove(res) : 저수준 문자(res)에서 올바른 문자를 리턴

GetOpType

- GetOpType(ea, n) : addres의 n번째 오퍼랜드 타입을 리턴

```
def GetOpType(ea, n):
    """
    Get type of instruction operand

    @param ea: linear address of instruction
    @param n: number of operand:
        0 - the first operand
        1 - the second operand

    @return: any of o_* constants or -1 on error
    """
    inslen = idaapi.decode_insn(ea)
    return -1 if inslen == 0 else idaapi.cmd.Operands[n].type
```

- idaapi.decode_insn(addr) : address를 디코드
- idaapi.cmd.Op1.type : 첫 번째 오퍼랜드 값의 타입
- idaapi.cmd.Op2.type : 두 번째 오퍼랜드 값의 타입

```
dism_addr = list(idautils.FuncItems(func))
for curr_addr in dism_addr:
    idaapi.decode_insn(curr_addr)
    if idaapi.cmd.Op1.type == idaapi.o_mem:
        print GetDisasm(curr_addr)
```

GetOpType

- o_void : 명령어에 오퍼랜드가 없으면 0을 리턴

```
Python>ea = here()
Python>op = GetOpType(ea, 0)
Python>if op == o_void:
Python> print GetDisasm(ea)
Python>
retn
```

- o_reg : 명령어에 오퍼랜드가 일반 레지스터이면 1을 리턴

```
Python>ea = here()
Python>op = GetOpType(ea, 0)
Python>if op == o_reg:
Python> print GetDisasm(ea)
Python>
mov     eax, [edx+44h]
```

- o_mem : 오퍼랜드가 직접적으로 메모리를 참조하면 2를 리턴하며, 주로 DATA를 참조하는데 사용

```
Python>ea = here()
Python>op = GetOpType(ea, 0)
Python>if op == o_mem:
Python> print GetDisasm(ea)
Python>
call    ds:WaitForSingleObject
```

```
0x401553 mov     dword_406798, eax
0x401560 mov     dword_4065B4, eax
0x401573 mov     dword_4065B8, 0
0x40157f call    ds:GetModuleHandleA
0x4016cf call    ds:MessageBoxA
0x401a05 call    ds:WaitForSingleObject
```

GetOpType

- o_phrase : 오퍼랜드가 인덱스 레지스터로 구성된 경우 3을 리턴

```
Python>ea = here()
Python>op = GetOpType(ea, 0)
Python>if op == o_phrase:
Python> print GetDisasm(ea)
Python>
mov     [esp+1Ch+var_1C], edx
```

```
0x401142 mov     [ecx], al
0x4011bf mov     [ecx], al
0x401296 movsd   [esp+80h+var_80], xmm0
0x4033ca mov     byte ptr [edx+eax], 0
0x4041d6 mov     [ecx], ax
0x4041f9 mov     dword ptr [ecx], 28h
0x4047c6 mov     byte ptr [edx+eax], 0
0x407095 mov     [eax], dl
0x407144 mov     [edx], ecx
0x407213 mov     [ecx], al
0x407301 mov     [edx], eax
```

- o_displ : 오퍼랜드가 레지스터와 변위 값(0x18과 같은 정수 값)으로 구성된 경우 4를 리턴

```
Python>ea = here()
Python>op = GetOpType(ea, 0)
Python>if op == o_displ:
Python> print GetDisasm(ea)
Python>
mov     [esi+4], edx
```

```
0x42a66d mov     [ebp+var_4], eax
0x42a6c2 test    byte ptr [eax+88h], 1
0x42a6d0 test    byte ptr [eax+58h], 1
0x42a71f mov     byte ptr [ecx+edx+28h], 0
0x42a749 mov     [eax+4], ecx
0x42a74f mov     [eax+8], ecx
0x42a755 or      dword ptr [eax+10h], 0FFFFFFFFh
0x42a75c mov     [eax+14h], ecx
0x42a762 mov     [eax+18h], ecx
0x42a768 mov     [eax+1Ch], ecx
```

GetOpType

- o_imm : 두 번째 오퍼랜드의 값이 정수와 같은 값이면 5를 리턴

```
Python>ea = here()
Python>op = GetOpType(ea, 1)
Python>if op == o_imm:
Python> print GetDisasm(ea)
Python>
shr     ebx, 1
```

```
0x42ad30 cmp     ax, 7Fh
0x42ad43 sub     esp, 20h
0x42ad46 and     esp, 0FFFFFFF0h
0x42ad6f xor     ecx, 80000000h
0x42ad75 add     ecx, 7FFFFFFFh
0x42ad7b adc     eax, 0
0x42ad82 adc     edx, 0
0x42ad8d add     ecx, 7FFFFFFFh
0x42ad93 sbb     eax, 0
0x42ad9a sbb     edx, 0
0x42ada3 test    edx, 7FFFFFFFh
0x42adc4 cmp     dword_43E0B8, 1
0x42addc shl     edx, 8
```

IDAPython – Sample Script(2)

- 모든 함수의 목록을 가져와 레지스터 타입(o_reg)의 레지스터(call, jmp)를 불러오는 스크립트

```
for func in idautils.Functions(): # 알려진 모든 함수의 목록을 가져옴
 flags = idc.GetFunctionFlags(func)
 if flags & FUNC_LIB or flags & FUNC_THUNK:
  continue
 dism_addr = list(idautils.FuncItems(func)) 함수 내에서 모든 주소를 가져온다.
 for line in dism_addr:
  m = idc.GetMnem(line)
  # mnemonic이 call이거나 jump인 경우 GetOpType 호출하여 피연산자 유형을 얻는다.
  if m == 'call' or m == 'jump':
   op = idc.GetOpType(line, 0)
   if op == o_reg: # 레지스터인지 확인
    print "0x%x %s" % (line, idc.GetDisasm(line))
```

```
0x4066c5 call    edx
0x40720b call    eax
0x4074ad call    eax
0x4075ad call    eax
0x4076ad call    eax
0x408443 call    eax
0x408843 call    edx
0x408fbc call    eax
0x409132 call    eax
0x40914f call    edx
```

Names

- Names : IDB 내부에 이름이 있는 주소를 iterator 형태로 리턴

```
Python>list(idautils.Names())
[(4198870L, '_WinMain@16'), (4201110L, '__get_startup_argv_mode'), (4201438L,
'unknown_libname_1'), (4201448L, 'unknown_libname_2'), (4202352L, '_memset'),
(4202387L, 'adjust_loop'), (4202393L, 'dwords'), (4202419L, 'main_loop_tail'),
(4202423L, 'tail'), (4202429L, 'finish'), (4202435L, 'toend'), (4202448L,
'_memcpy'), (4202560L, 'LeadUp1'), (4202604L, 'LeadUp2'), (4202640L, 'LeadUp3'),
(4202700L, 'UnwindUp7'), (4202708L, 'UnwindUp6'), (4202716L, 'UnwindUp5'),
(4202724L, 'UnwindUp4'), (4202732L, 'UnwindUp3'), (4202740L, 'UnwindUp2'),
(4202748L, 'UnwindUp1'), (4202767L, 'UnwindUp0'), (4202792L, 'TrailUp0'), (4202800L,
'TrailUp1'), (4202812L, 'TrailUp2'), (4202832L, 'TrailUp3'), (4202952L,
'LeadDown1'), (4202984L, 'LeadDown2'), (4203024L, 'LeadDown3'), (4203108L,
'UnwindDown7'), (4203116L, 'UnwindDown6'), (4203124L, 'UnwindDown5'), (4203132L,
'UnwindDown4'), (4203140L, 'UnwindDown3'), (4203148L, 'UnwindDown2'), (4203156L,
```

CodeRefsTo

- CodeRefsTo(address, flags) : addres의 상호참조 값을 iterator 형태로 리턴

```
def CodeRefsTo(ea, flow):
    """
    Get a list of code references to 'ea'

    @param ea:   Target address
    @param flow: Follow normal code flow or not
    @type  flow: Boolean (0/1, False/True)

    @return: list of references (may be empty list)

    Example::

        for ref in CodeRefsTo(ScreenEA(), 1):
            print ref
    """
    if flow == 1:
        return refs(ea, idaapi.get_first_cref_to, idaapi.get_next_cref_to)
    else:
        return refs(ea, idaapi.get_first_fcref_to, idaapi.get_next_fcref_to)
```

CodeRefsTo

- IDA에서 함수의 이름을 커서에 대고 단축 키(X)를 누르면 상호참조하는 주소를 확인할 수 있음

xrefs to sub_401D7B

Direction	Typ	Address	Text
Up	p	WinMain(x,x,x,x)+25	call sub_401D7B
Up	p	WinMain(x,x,x,x)+51	call sub_401D7B
Up	p	WinMain(x,x,x,x)+6F	call sub_401D7B
	p	WinMain(x,x,x,x)+C9	call sub_401D7B
Do...	p	WinMain(x,x,x,x)+105	call sub_401D7B
Do...	p	WinMain(x,x,x,x)+136	call sub_401D7B
Do...	p	WinMain(x,x,x,x)+154	call sub_401D7B
Do...	p	WinMain(x,x,x,x)+18F	call sub_401D7B
Do...	p	WinMain(x,x,x,x)+1AC	call sub_401D7B
Do...	p	sub_4013CD+C	call sub_401D7B
Do...	p	sub_4013CD+2C	call sub_401D7B
Do...	p	sub_4013CD+4C	call sub_401D7B
Do...	p	sub_4013CD+6C	call sub_401D7B
Do...	p	sub_4014F2+A9	call sub_401D7B
Do...	p	sub_4014F2+17D	call sub_401D7B

```
Python>map(hex, list(CodeRefsTo(0x401D7B, 0)))
['0x4011fbL', '0x401227L', '0x401245L', '0x40129fL', '0x4012dbL',
'0x40130cL', '0x40132aL', '0x401365L', '0x401382L', '0x4013d9L',
'0x4013f9L', '0x401419L', '0x401439L', '0x40159bL', '0x40166fL']
```

IDAPython – Sample Script(3)

- 특정 검색하고 싶은 함수의 상호참조되는 주소를 CodeRefsTo 함수를 통해 검색 후 해당 주소를 빨간색으로 변환하는 스크립트
- LocByName(name) : 프로그램 바이트인 이름(name)의 주소를 리턴(API 주소 찾을 시 유용)

```
from idaapi import *
#danger_funcs = ["strcpy", "sprintf", "strncpy"]
danger_funcs = ["HeapAlloc", "VirtualAlloc", "WriteFile", "LoadLibrary", "GetProcAddrss"]
for func in danger_funcs:

    addr = LocByName(func) # API에 대한 주소를 가져옴
    if addr != BADADDR:
        # Xref to(상호 참조) 기능을 통해 API를 상호 참조하는 주소를 리스트 형태로 리턴
        cross_ref = CodeRefsTo(addr, 0)
        print "Cross References to %s" % func
        print "------------------------------"

        for ref in cross_ref:
            print "%08x" % ref
            SetColor(ref, CIC_ITEM, 0x0000ff) # 찾은 API의 주소를 빨간색으로 변환
```

CodeRefsTo

- GetProcAddress가 사용되는 곳의 주소를 빨간색으로 변환 후 쉽게 식별 가능

```
.text:00403F58                push    offset ProcName ; "MessageBoxA"
.text:00403F5D                push    edi             ; hModule
.text:00403F5E                call    esi ; GetProcAddress
.text:00403F60                test    eax, eax
.text:00403F62                mov     dword_406728, eax
.text:00403F67                jz      short loc_403FB9
.text:00403F69                push    offset aGetactivewindo ; "GetActiveWindow"
.text:00403F6E                push    edi             ; hModule
.text:00403F6F                call    esi ; GetProcAddress
.text:00403F71                push    offset aGetlastactivep ; "GetLastActivePopup"
.text:00403F76                push    edi             ; hModule
.text:00403F77                mov     dword_40672C, eax
.text:00403F7C                call    esi ; GetProcAddress
.text:00403F7E                mov     dword_406730, eax
```

CodeRefsFrom

- CodeRefsFrom(address, flags) : address가 정의된 곳을 참조

```
def CodeRefsFrom(ea, flow):
    """
    Get a list of code references from 'ea'

    @param ea:   Target address
    @param flow: Follow normal code flow or not
    @type  flow: Boolean (0/1, False/True)

    @return: list of references (may be empty list)

    Example::

        for ref in CodeRefsFrom(ScreenEA(), 1):
            print ref
    """
    if flow == 1:
        return refs(ea, idaapi.get_first_cref_from, idaapi.get_next_cref_from)
    else:
        return refs(ea, idaapi.get_first_fcref_from, idaapi.get_next_fcref_from)
```

CodeRefsFrom

- CodeRefsTo 함수와 반대로 참조하는 값이 아닌 정의된 값을 리턴

```
.text:0040364D                add     esi, 0Fh
.text:00403650                and     esi, 0FFFFFFF0h
.text:00403653                push    esi             ; dwBytes
.text:00403654                push    0               ; dwFlags
.text:00403656                push    hHeap           ; hHeap
.text:0040365C                call    ds:HeapAlloc
```

```
Python>addr = here()
Python>map(hex, list(CodeRefsFrom(here(), 0)))
['0x405000L']
```

```
.idata:00405000 ; LPVOID __stdcall HeapAlloc(HANDLE hHeap, DWORD dwFlags, SIZE_T dwBytes)
.idata:00405000                 extrn HeapAlloc:dword   ; CODE XREF: sub_401047+18↑p
.idata:00405000                                         ; __heap_alloc+2E↑p ...
```

XrefsTo

- XrefsTo : CodeRefsTo와 기능은 동일하지만 내부에 type 값을 확인 가능

```
for xref in idautils.XrefsTo(ea, 0):
 print xref.type, idautils.XrefTypeName(xref.type), hex(xref.frm), hex(xref.to), xref.iscode
```

```
17 Code_Near_Call 0x403097L 0x40506cL 1
3 Data_Read 0x403097L 0x40506cL 0
```

- xref.type : xrefs type value를 출력

```
0 = 'Data_Unknown'
1 = 'Data_Offset'
2 = 'Data_Write'
3 = 'Data_Read'
4 = 'Data_Text'
5 = 'Data_Informational'
16 = 'Code_Far_Call'
17 = 'Code_Near_Call'
18 = 'Code_Far_Jump'
19 = 'Code_Near_Jump'
20 = 'Code_User'
21 = 'Ordinary_Flow'
```

- xref.iscode : 참조된 주소가 코드 세그먼트에 존재하면 1 리턴

FindBinary

- FindBinary(ea, flag, searchstr, radix=16) : 시작할 address(ea), flag를 지정해 찾고 싶은 바이너리 값(searchstr)을 검색한다.

```
def FindBinary(ea, flag, searchstr, radix=16):
    """
    @param ea: start address
    @param flag: combination of SEARCH_* flags
    @param searchstr: a string as a user enters it for Search Text in Core
    @param radix: radix of the numbers (default=16)

    @return: ea of result or BADADDR if not found

    @note: Example: "41 42" - find 2 bytes 41h,42h (radix is 16)
    """
    endea = flag & 1 and idaapi.cvar.inf.maxEA or idaapi.cvar.inf.minEA
    return idaapi.find_binary(ea, endea, searchstr, radix, flag)
```

FindBinary

- Flag values

```
SEARCH_UP = 0
SEARCH_DOWN = 1
# UP, DOWN은 검색을 수행 할 방향을 선택하는 데 사용

SEARCH_NEXT = 2
# SEARCH_NEXT는 다음 개체를 가져오는데 사용

SEARCH_CASE = 4
# 대소문자 구분을 지정하는데 사용

SEARCH_REGEX = 8
# 정규표현식 사용

SEARCH_NOBRK = 16

SEARCH_NOSHOW = 32
# 검색 진행률을 표시하지 않는다.
```

```
opcode_str = "55 8B EC"  # PUSH EBP
binary_address = FindBinary(0, 1, opcode_str)
print hex(binary_address), GetDisasm(binary_address)
```

```
Python>
0x401000L push    ebp
```

```
.text:00401000                push    ebp
.text:00401001                mov     ebp, esp
.text:00401003                sub     esp, 8
```

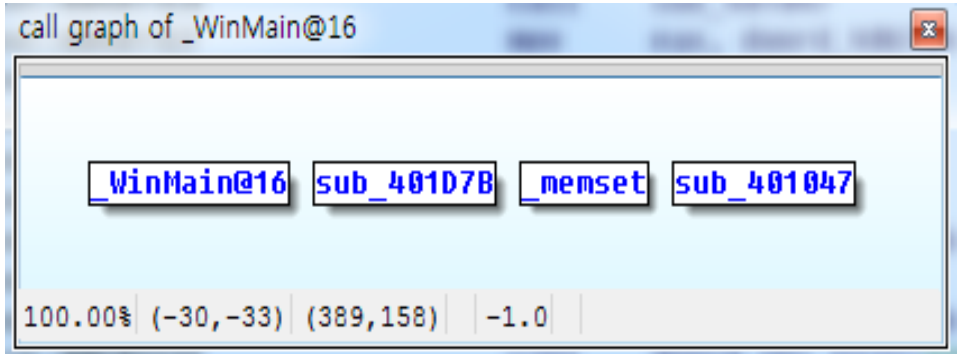
IDAPython – idaapi.GraphViewer(Graph)

Idaapi.GraphViewer 클래스를 사용해 그래프 형태로 스크립트 개발 가능

```
from idaapi import GraphViewer

class MyGraph(GraphViewer):
    def __init__(self, funcname, result):
        GraphViewer.__init__(self, "call graph of " + funcname)
        self.funcname = funcname
        self.result = result

    def OnRefresh(self):
        self.Clear()
        id = self.AddNode(self.funcname)
        for x in self.result.keys():
            callee = self.AddNode(x)
            d=self.AddEdge(id, callee)
            self.AddEdge(id, d)
        return True
```



IDAPython – idaapi.Choose2(GUI)

idaapi.Choose2 클래스를 사용해 GUI 형태로 스크립트 개발 가능

```python
from idautils import *
from idc import *
from idaapi import *

target_functions = [
    "VirtualAlloc",
    "WriteFile",
    "CreateThread",
    "send",
    "recv",
    "GetComputerName",
]

global func_lib
func_lib = {}

class Xref_to_Import_Function(Choose2):

    def __init__(self, title):
        Choose2.__init__(self, title, [ ["Address", 10 | Choose2.CHCOL_HEX],\
                                        ["Name", 20 | Choose2.CHCOL_PLAIN] ])
        self.title = title
        self.n = 0
        self.icon = 41
        self.write_import_api()

    def imp_cb(self, ea, name, ord):
        func_lib[name] = hex(ea)

        #print "%08x: %s" % (ea, name)

        return True
```

Address	Name
0x00403dfe	VirtualAlloc
0x00403e8a	VirtualAlloc
0x00403097	WriteFile

```
.text:00403077 ; ---------------------------------------------------------------------------
.text:00403077
.text:00403077 loc_403077:                             ; CODE XREF: __NMSG_WRITE+3C↑j
.text:00403077                                         ; __NMSG_WRITE+4D↑j
.text:00403077                 lea     eax, [ebp+NumberOfBytesWritten]
.text:0040307A                 lea     esi, off_406204[esi]
.text:00403080                 push    0               ; lpOverlapped
.text:00403082                 push    eax             ; lpNumberOfBytesWritten
.text:00403083                 push    dword ptr [esi] ; char *
.text:00403085                 call    _strlen
.text:0040308A                 pop     ecx
.text:0040308B                 push    eax             ; nNumberOfBytesToWrite
.text:0040308C                 push    dword ptr [esi] ; lpBuffer
.text:0040308E                 push    0FFFFFFF4h      ; nStdHandle
.text:00403090                 call    ds:GetStdHandle
.text:00403096                 push    eax             ; hFile
.text:00403097                 call    ds:WriteFile
```

IDAPython Plugin - Bindiff

IDA Bindiff 플러그인(IDB 비교)

Matched Functions

similarity	confide	change	EA primary	name primary	EA secondary	name secondary
0.85	0.92	-I--E--	0000031F	sub_31F_10	00002980	sub_2980_59
0.72	0.78	-I--E--	00000212	sub_212_4	0000527E	sub_527E_101
0.72	0.78	-I--E--	00001647	sub_1647_31	00004210	sub_4210_74
0.40	0.62	-I--E--	0000F51F	sub_F51F_41	00005A58	sub_5A58_120
0.34	0.50	GI--E--	000000F0	sub_F0_1	000028A0	sub_28A0_58
0.19	0.31	GI--EL-	000004CF	sub_4CF_13	0000556C	sub_556C_115
0.17	0.24	GI--E--	0000161E	sub_161E_30	00002BE8	sub_2BE8_60
0.17	0.23	GI--EL-	00000E5A	sub_E5A_21	00003960	sub_3960_72
0.14	0.25	GI--EL-	00000FCC	sub_FCC_22	00001D2C	sub_1D2C_49
0.12	0.17	GI--EL-	00000DCF	sub_DCF_18	000054B0	sub_54B0_112
0.10	0.17	GI--E--	00000078	sub_78_0	00001F90	sub_1F90_51
0.07	0.18	GI--EL-	0000155B	sub_155B_29	00006048	j_j_free
0.07	0.12	GI--EL-	000010D5	sub_10D5_25	00001E78	sub_1E78_50
0.02	0.06	GI--EL-	00001958	sub_1958_38	00001AD4	sub_1AD4_48
0.02	0.03	GI--E--	000019DC	sub_19DC_39	00005FA8	j_j_strerror
0.01	0.03	GI--E--	00000168	sub_168_2	000036A4	sub_36A4_70
0.01	0.02	GI--E--	0000192D	sub_192D_37	00006098	j_j_ptrace
0.01	0.02	GI--E--	000018C4	sub_18C4_36	00006018	j_j_fwrite
0.01	0.03	GI--EL-	00000225	sub_225_5	00003B3C	sub_3B3C_73
0.01	0.02	GI--E--	000017B5	sub_17B5_34	00005FB8	j_j_fprintf
0.01	0.02	GI--EL-	0000104F	sub_104F_23	00005FE8	j_j_printf
0.01	0.02	GI--E--	000002A7	sub_2A7_7	00006488	j_j___cxa_type_match
0.01	0.02	GI--E--	0000025E	sub_25E_6	00005FC8	j_j_puts
0.01	0.02	GI--E--	000002D6	sub_2D6_8	00006478	j_j___cxa_begin_clean

Statistics

Primary Unmatched

EA	Name	Basicblock
00000C8A	sub_C8A_14	1
00000CB6	sub_CB6_15	13
00000DA0	sub_DA0_17	4
00000E1B	sub_E1B_19	3
00001094	sub_1094_24	5
000012E3	sub_12E3_27	9
0000168A	sub_168A_32	5
0000F52B	sub_F52B_42	3

IDAPython 활용

IDAPython – Sample Script(4)

```
signed int sub_401000()
{
  HMODULE v0; // eax@1
  HMODULE v1; // eax@38
  HMODULE v2; // eax@48
  HMODULE v3; // eax@56
  HMODULE v4; // eax@60
  HMODULE v5; // eax@66
  signed int result; // eax@69

  v0 = LoadLibraryA(LibFileName);
  hModule = v0;
  if ( !v0 )
    goto LABEL_73;
  dword_40C374 = (int)GetProcAddress(v0, ProcName);
  if ( !dword_40C374 )
    goto LABEL_73;
  dword_40C378 = (int)GetProcAddress(hModule, aCreatefilew);
  if ( !dword_40C378 )
    goto LABEL_73;
  dword_40C3C8 = (int)GetProcAddress(hModule, aSleep);
  if ( !dword_40C3C8 )
    goto LABEL_73;
  dword_40C384 = (int)GetProcAddress(hModule, aWritefile);
  if ( !dword_40C384 )
    goto LABEL_73;
  dword_40C36C = (int)GetProcAddress(hModule, aDeletefilea);
  if ( !dword_40C36C )
    goto LABEL_73;
  dword_40C370 = (int)GetProcAddress(hModule, aDeletefilew);
  if ( !dword_40C370 )
    goto LABEL_73;
  dword_40C37C = (int)GetProcAddress(hModule, aCopyfilea);
  if ( !dword_40C37C )
    goto LABEL_73;
```

dword_xxxx..?????????????

```
result = sub_401000();
if ( result )
{
  sub_402EC0(aSedebugprivile);
  memset(&v12, 0, 0x104u);
  ((void (__cdecl *)(_DWORD, int *, signed int, _DWORD))dword_40C3D0)(0, &v12, 26, 0);
  dword_40C338(&v11, aMicrosoft);
  dword_40C3DC(&v10, 0);
  dword_40C338(&v9, aHnc);
  dword_40C3DC(&v8, 0);
  dword_40C3D4(&v7, &v7, 260);
  dword_40C348(byte_40C114, aSS, &v6, a1_hwp);
  dword_40C348(&unk_40C218, aSS, &v6, aHupdate_ex);
  v15[0] = 0;
  memset(&v15[1], 0, 0x100u);
  v16 = 0;
  v17 = 0;
  dword_40C3D0(0, v15, 7, 0, v5);
  dword_40C338(&v14, aViso_exe);
  sub_402BE0(&v13);
  sub_402950();
  sub_402A10();
}
return result;
```

IDAPython – Sample Script(4)

0x401000 함수의 디스어셈 코드

```
mov     ecx, hModule
push    offset aSleep           ; "Sleep"
push    ecx                     ; hModule
call    esi ; GetProcAddress
test    eax, eax
mov     dword_40C3C8, eax
jz      loc_4016C9
mov     edx, hModule
push    offset aWritefile       ; "WriteFile"
push    edx                     ; hModule
call    esi ; GetProcAddress
test    eax, eax
mov     dword_40C384, eax
jz      loc_4016C9
mov     eax, hModule
push    offset aDeletefilea     ; "DeleteFileA"
push    eax                     ; hModule
call    esi ; GetProcAddress
test    eax, eax
mov     dword_40C36C, eax
jz      loc_4016C9
```

데이터 영역 내부의 문자열

```
; CHAR aWritefile[]
aWritefile      db 'WriteFile',0
                align 10h
; CHAR aSleep[]
aSleep          db 'Sleep',0
                align 4
; CHAR aCreatefilew[]
aCreatefilew    db 'CreateFileW',0
; CHAR ProcName[]
ProcName        db 'CreateFileA',0
```

IDAPython – Sample Script(4)

- 주소 이름(dword_xxxx)을 참조되는 데이터 영역 문자열로 변경해주는 스크립트

```
from idautils import *
from idaapi import *
from idc import *

func = 0x401000

# func(401000) 함수의 모든 주소를 iterator 형태로 리턴
dism_addr = idautils.FuncItems(func)
for line in dism_addr:
 op = GetOpType(line, 0)
 if op == o_mem: # 젓 번째 오퍼랜드 타입이 o_mem 인지 꽉인
                 # ex) dword_xxxx, eax

  # o_mem 타입의 주소라면, 젓 번째 오퍼랜드의 값이 들어있는 주소를 리턴
  target_address = GetOperandValue(line, 0)

  while True:
   line = idc.PrevHead(line) # 이전 명령 주소 리턴

   # 명령 주소의 레지스터가 PUSH 이고 젓 번째 오퍼랜드의 값이 offset인 것을 탐색
   if GetMnem(line) == "push" and "offset" in GetOpnd(line, 0):
    str_address = GetOperandValue(line, 0) # 변경할 문자가 존재하는 주소 리턴
    str = GetString(str_address, -1, ASCSTR_C) # 주소에 존재하는 문자열을 가져옴
    MakeName(target_address, 'a__'+str) # 잦은 문자열로 변경
    break
```

IDAPython – Sample Script(4)

```
signed int sub_401000()
{
  HMODULE v0; // eax@1
  HMODULE v1; // eax@38
  HMODULE v2; // eax@48
  HMODULE v3; // eax@56
  HMODULE v4; // eax@60
  HMODULE v5; // eax@66
  signed int result; // eax@69

  v0 = LoadLibraryA(LibFileName);
  a__Kernel32_dll = v0;
  if ( !v0 )
    goto LABEL_73;
  a__CreateFileA = (int)GetProcAddress(v0, ProcName);
  if ( !a__CreateFileA )
    goto LABEL_73;
  a__CreateFileW = (int)GetProcAddress(a__Kernel32_dll, aCreatefilew);
  if ( !a__CreateFileW )
    goto LABEL_73;
  a__Sleep = (int (__stdcall *)(_DWORD))GetProcAddress(a__Kernel32_dll, aSleep);
  if ( !a__Sleep )
    goto LABEL_73;
  a__WriteFile = (int)GetProcAddress(a__Kernel32_dll, aWritefile);
  if ( !a__WriteFile )
    goto LABEL_73;
  a__DeleteFileA = (int)GetProcAddress(a__Kernel32_dll, aDeletefilea);
  if ( !a__DeleteFileA )
    goto LABEL_73;
  a__DeleteFileW = (int)GetProcAddress(a__Kernel32_dll, aDeletefilew);
  if ( !a__DeleteFileW )
    goto LABEL_73;
```

```
result = sub_401000();
if ( result )
{
  sub_402EC0((int)aSedebugprivile);
  memset(&v12, 0, 0x104u);
  ((void (__cdecl *)(_DWORD, int *, signed int, _DWORD))a__SHGetSpecialFolderPathA)(0, &v12, 26, 0);
  a__lstrcatA(&v11, aMicrosoft);
  a__CreateDirectoryA(&v10, 0);
  a__lstrcatA(&v9, aHnc);
  a__CreateDirectoryA(&v8, 0);
  a__GetShortPathNameA(&v7, &v7, 260);
  a__wsprintfA(byte_40C114, aSS, &v6, a1_hwp);
  a__wsprintfA(&unk_40C218, aSS, &v6, aHupdate_ex);
  v15[0] = 0;
  memset(&v15[1], 0, 0x100u);
  v16 = 0;
  v17 = 0;
```

스크립트 적용 결과 코드가 가시적으로 보여짐

ShadowPad Malware Sample

Xshell(ShadowPad 악성코드)

- 네트워크 연결 및 처리를 담당하는 필수 DLL인 nssock2.dll이 변조

```
void *__thiscall sub_1000C6C0(void *this)
{
  void *v2; // [sp+0h] [bp-18h]@1
  LPVOID v3; // [sp+8h] [bp-10h]@1
  unsigned int i; // [sp+10h] [bp-8h]@1
  unsigned int v5; // [sp+14h] [bp-4h]@1

  v2 = this;
  v3 = VirtualAlloc(0, 0xFB48u, 0x1000u, 0x40u);
  v5 = unk_1000F718;
  for ( i = 0; i < 0xFB44; ++i )
  {
    *((_BYTE *)v3 + i) = v5 ^ *((_BYTE *)&unk_1000F718 + i + 4);
    v5 = -910240943 * ((v5 >> 16) + (v5 << 16)) - 1470258743;
  }
  if ( (unsigned int)((int (__stdcall *)(_DWORD))v3)(0) < 0x1000 )
    MessageBoxA(0, "###ERROR###", 0, 0);
  return v2;
}
```

nssock2.dll – 문제의 1000C6C0 함수

ShadowPad Malware Sample

Xshell(ShadowPad 악성코드)

- 네트워크 연결 및 처리를 담당하는 필수 DLL인 nssock2.dll이 변조

```
void *__thiscall sub_1000C6C0(void *this)
{
  void *v2; // [sp+0h] [bp-18h]@1
  LPVOID v3; // [sp+8h] [bp-10h]@1
  unsigned int i; // [sp+10h] [bp-8h]@1
  unsigned int v5; // [sp+14h] [bp-4h]@1

  v2 = this;
  v3 = VirtualAlloc(0, 0xFB48u, 0x1000u, 0x40u);
  v5 = unk_1000F718;
  for ( i = 0; i < 0xFB44; ++i )
  {
    *((_BYTE *)v3 + i) = v5 ^ *((_BYTE *)&unk_1000F718 + i + 4);
    v5 = -910240943 * ((v5 >> 16) + (v5 << 16)) - 1470258743;
  }
  if ( (unsigned int)((int (__stdcall *)(_DWORD))v3)(0) < 0x1000 )
    MessageBoxA(0, "###ERROR###", 0, 0);
  return v2;
}
```

```
.rdata:1000F714                 dd 1F04Ch
.rdata:1000F718 unk_1000F718    db    4
.rdata:1000F719                 db 0F2h ;
.rdata:1000F71A                 db  56h ; V
.rdata:1000F71B                 db 0CFh ;
.rdata:1000F71C                 db  8Fh ;
.rdata:1000F71D                 db 0B3h ;
.rdata:1000F71E                 db 0F0h ;
.rdata:1000F71F                 db  0Ah
.rdata:1000F720                 db 0F2h ;
.rdata:1000F721                 db  1Dh
.rdata:1000F722                 db  95h ;
.rdata:1000F723                 db 0DFh ;
.rdata:1000F724                 db  7Dh ; }
.rdata:1000F725                 db  7Bh ; {
.rdata:1000F726                 db 0E4h ;
.rdata:1000F727                 db 0B0h ;
.rdata:1000F728                 db  0Eh
```

```
1000F710  4C 04 02 00 4C F0 01 00  04 F2 56 CF 8F B3 F0 0A  L...L.....V.....
1000F720  F2 1D 95 DF 7D 7B E4 B0  0E F9 E1 B2 20 8B 9D 5C  ....}{...... ..W
1000F730  9B 79 CD 03 12 C3 89 6E  01 59 F0 5B A7 E5 84 0D  .y.....n.Y.[....
1000F740  4A BF 00 FA 1E 7E B1 49  67 1E 61 C6 DE A9 A8 45  J....~.Ig.a....E
1000F750  47 1B E5 6A 90 7D 60 0D  C4 91 F6 08 9C 4F 94 5B  G..j.}`......O.[
1000F760  F8 F6 04 34 23 09 68 F6  8E E7 5D E9 78 9F 59 E1  ...4#.h...].x.Y.
1000F770  E1 5F DE 60 00 3C AE 4B  C7 BA A7 9F 1E E1 79 EC  ._.`.<.K......y.
1000F780  D4 E9 7E A8 96 3B 17 B8  15 09 74 E0 DF 7E A7 9C  ..~..;....t..~..
1000F790  26 53 ED F8 15 C5 7D 13  54 18 6F E8 52 2D 01 A9  &S....}.T.o.R-..
1000F7A0  70 42 F2 F7 E2 00 1C A6  D5 0D B9 AF AF 4A 73 D8  pB...........Js.
1000F7B0  BF 10 C3 F8 D6 57 F1 D2  5F CA 72 57 11 5F 1C AE  .....W.._.rW._..
1000F7C0  4B 32 99 21 F1 3B E0 1A  CB A0 75 4A 11 D4 55 B6  K2.!.;....uJ..U.
1000F7D0  61 62 95 21 B0 41 56 BB  F0 E2 49 DB B7 D8 B0 B4  ab.!.AV...I.....
```

nssock2.dll – 문제의 1000C6C0 함수

IDAPython – Sample Script(5)

- 1000F718 주소의 XOR 인코딩된 바이너리를 덤프하는 스크립트

```
from idaapi import *
from idc import *
from idautils import *
from binascii import unhexlify

start = 0x1000F718 # 쉘코드 시작 주소
end = idc.NextHead(start) # 쉘코드 시작 주소의 끝을 알아내기 위해
                          # 다음 헤드의 젓 주소를 알아냄

with open('encoded_shellcode.bin', 'wb') as shellcode:
    while(start < end): # 쉘코드가 포함되어 있는 주소를 끝날 때까지 반복
        int_data = idc.IdbByte(start) # 주소의 헥스 값을 불러옴
        try:
            unhex_byte = unhexlify('%x' % int_data)
        except:
            unhex_byte = unhexlify('0%x' % int_data)
        shellcode.write(unhex_byte) # 인코딩된 쉘코드 작성
        start = idc.NextAddr(start)

print 'dump to encoded shellcode'
```

XOR 디코딩(0x1000C6C0) 스크립트

```
void *__thiscall sub_1000C6C0(void *this)
{
  void *v2; // [sp+0h] [bp-18h]@1
  LPVOID v3; // [sp+8h] [bp-10h]@1
  unsigned int i; // [sp+10h] [bp-8h]@1
  unsigned int v5; // [sp+14h] [bp-4h]@1

  v2 = this;
  v3 = VirtualAlloc(0, 0xFB48u, 0x1000u, 0x40u);
  v5 = unk_1000F718;
  for ( i = 0; i < 0xFB44; ++i )
  {
    *((_BYTE *)v3 + i) = v5 ^ *((_BYTE *)&unk_1000F718 + i + 4);
    v5 = -910240943 * ((v5 >> 16) + (v5 << 16)) - 1470258743;
  }
  if ( (unsigned int)((int (__stdcall *)(_DWORD))v3)(0) < 0x1000 )
    MessageBoxA(0, "###ERROR###", 0, 0);
  return v2;
}
```

```
with open('encoded_shellcode.bin', 'rb') as shellcode2:
    data=shellcode2.read()
    (xor_key,) = struct.unpack("<L", data[0:4])

    decoded_bytes=''
    for ch in data[4:]:
        decoded_bytes += chr(ord(ch) ^ (xor_key & 0xff))
        xor_key = ((xor_key << 0x10)  + (xor_key >> 0x10)) & 0xffffffff
        xor_key = ((xor_key * 0xC9BED351) - 0x57A25E37) & 0xffffffff

    with open('decoded_shellcode.bin', 'wb') as shellcode3:
        shellcode3.write(decoded_bytes)

print 'dump to decoded shellcode'
```

ShadowPad Malware Sample

어셈블리 코드 중간에 쓰레기 값(E9)을 추가해서 디스어셈블을 망가뜨리는 트릭

```
seg000:0000F52B ; ---------------------------------------------------------------------------
seg000:0000F52B
seg000:0000F52B loc_F52B:                               ; CODE XREF: sub_F51F+1↑p
seg000:0000F52B                 push    ebp
seg000:0000F52C                 mov     ebp, esp
seg000:0000F52E                 sub     esp, 434h
seg000:0000F534                 push    ebx
seg000:0000F535                 push    esi
seg000:0000F536                 push    edi
seg000:0000F537                 js      short near ptr loc_F53B+1
seg000:0000F539                 jns     short near ptr loc_F53B+1
seg000:0000F53B
seg000:0000F53B loc_F53B:                               ; CODE XREF
seg000:0000F53B                                         ; seg000:00
seg000:0000F53B                 jmp     near ptr 3196A4h
seg000:0000F53B ; ---------------------------------------------------------------------------
```

```
0000F520  E8 06 00 00 00 89 EC 5D  C2 04 00 55 8B EC 81 EC  .......]...U....
0000F530  34 04 00 00 53 56 57 78  03 79 01 E9 64 A1 30 00  4...SVWx.y..d.0.
0000F540  00 00 8B 40 0C 8B 58 0C  33 FF EB 3C 8B 73 30 89  ...@..X.3..<.s0.
0000F550  7D F4 66 39 3E 74 2A 78  03 79 01 E9 0F B6 0E 8B  }.f9>t*x.y......
0000F560  45 F4 C1 C8 08 83 C9 20  03 C1 35 A3 D9 35 7C 83  E...... ..5..5|.
0000F570  C6 02 89 45 F4 66 39 3E  75 DD 3D 61 12 5B FD 74  ...E.f9>u.=a.[.t
0000F580  0E 7C 03 7D 01 E9 8B 1B  39 7B 18 75 BF EB 07 8B  .|.}....9{.u....
0000F590  5B 18 3B DF 75 08 33 C0  40 E9 9F 05 00 00 8B 43  [.;.u.3.@......C
0000F5A0  3C 8B 74 18 78 89 7D D8  89 7D E0 89 7D FC 89 7D  <.t.x.}..}..}..}
0000F5B0  DC 03 F3 70 03 71 01 E9  8B 46 20 03 C3 89 45 EC  ...p.q...F ...E.
```

ShadowPad Malware Sample

수동으로 undefined(단축키 U) + code(단축키 C)로 변환하는 과정을 수행한 결과, 동일한 코드 발견

```
seg000:0000F52B ; ---------------------------------------------------------------------------
seg000:0000F52B
seg000:0000F52B loc_F52B:                               ; CODE XREF: sub_F51F+1↑p
seg000:0000F52B                 push    ebp
seg000:0000F52C                 mov     ebp, esp
seg000:0000F52E                 sub     esp, 434h
seg000:0000F534                 push    ebx
seg000:0000F535                 push    esi
seg000:0000F536                 push    edi
seg000:0000F537                 js      short loc_F53C
seg000:0000F539                 jns     short loc_F53C
seg000:0000F539 ; ---------------------------------------------------------------------------
seg000:0000F53B                 db 0E9h ;
seg000:0000F53C ; ---------------------------------------------------------------------------
seg000:0000F53C
seg000:0000F53C loc_F53C:                               ; CODE XREF: seg000:0000F537↑j
seg000:0000F53C                                         ; seg000:0000F539↑j
seg000:0000F53C                 mov     eax, fs:dword_30
seg000:0000F542                 mov     eax, [eax+0Ch]
seg000:0000F545                 mov     ebx, [eax+0Ch]
seg000:0000F548                 xor     edi, edi
seg000:0000F54A                 jmp     short loc_F588
seg000:0000F54C ; ---------------------------------------------------------------------------
seg000:0000F54C
seg000:0000F54C loc_F54C:                               ; CODE XREF: seg000:0000F58B↓j
seg000:0000F54C                 mov     esi, [ebx+30h]
seg000:0000F54F                 mov     [ebp-0Ch], edi
seg000:0000F552                 cmp     [esi], di
seg000:0000F555                 jz      short loc_F581
seg000:0000F557                 js      short near ptr loc_F55B+1
seg000:0000F559                 jns     short near ptr loc_F55B+1
seg000:0000F55B
seg000:0000F55B loc_F55B:                               ; CODE XREF: seg000:0000F557↑j
seg000:0000F55B                                         ; seg000:0000F559↑j
seg000:0000F55B                 jmp     near ptr 8B0FAB6Fh
```

IDAPython – Sample Script(6)

반복되는 난독화 과정을 스크립트로 풀기 위해 위 코드에서 사용되는 jump condition 정보를 모두 수집

```
70 = jo
71 = jno

73 = jnb, jnc, jae
72 = jb, jc, jnae

74 = je, jz
75 = jne, jnz

77 = ja
76 = jbe

78 = js
79 = jns

7B = jnp, jpo
7A = jp, jpe

7C = jl
7D = jge

7F = jg
7E = jle
```

- Jump conditio의 hex 값은 70~7F로 확인
- 7x 03 7x 01 E8|E9 형태로 반복되어 경우의 수를 모두 리스트에 작성

```
import idaapi
import idc
import idautils

start_addr = MinEA()
end_addr = MaxEA()

# 조건 점프 명령어 뒷 부분에 쓰레기 값(E9==jmp, E8==CALL)을 합친 리스트
jump_condition_pattern = ['70 03 71 01 E9', '70 03 71 01 E8', '71 03 70 01 E9', '71 03 70 01 E8',\
                          '73 03 72 01 E9', '73 03 72 01 E8', '72 03 73 01 E9', '72 03 73 01 E8',\
                          '74 03 75 01 E9', '74 03 75 01 E8', '75 03 74 01 E9', '75 03 74 01 E8',\
                          '77 03 76 01 E9', '77 03 76 01 E8', '76 03 77 01 E9', '76 03 77 01 E8',\
                          '78 03 79 01 E9', '78 03 79 01 E8', '79 03 78 01 E9', '79 03 78 01 E8',\
                          '7B 03 7A 01 E9', '7B 03 7A 01 E8', '7A 03 7B 01 E9', '7A 03 7B 01 E8',\
                          '7C 03 7D 01 E9', '7C 03 7D 01 E8', '7D 03 7C 01 E9', '7D 03 7C 01 E8',\
                          '7F 03 7E 01 E9', '7F 03 7E 01 E8', '7E 03 7F 01 E9', '7E 03 7F 01 E8']
```

IDAPython – Sample Script(6)

0x0부터 순차적으로 jump condition이 존재하는 리스트의 각 요소를 검색 후 리스트에 저장

```
def custom_finding_binary(opcode_str):
    ret = []
    if len(opcode_str) > 1: # 다수의 값을 찾을 경우
        for index_opcode_str in opcode_str:
            # jump_condition_pattern 리스트에 존재하는 값을 0x0부터 순차적으로 검색(SEARCH_DOWN)
            ea = idc.FindBinary(0, SEARCH_DOWN, index_opcode_str)
            while ea != idc.BADADDR: # 값이 BADADDR(0xFFFFFFFF)이 아닐 때까지
                ret.append(ea) # 찾은 값을 리스트에 저장
                try:
                    # 순차적으로 다시 검색
                    ea = idc.FindBinary(ea + 4, 1, index_opcode_str)
                except:
                    pass
        return ret
    else: # 한 개의 값을 찾을 경우
        ea = idc.FindBinary(0, 1, opcode_str)
        while ea != idc.BADADDR:
            ret.append(ea)
            ea = idc.FindBinary(ea + 4, 1, opcode_str)
        return ret
```

IDAPython – Sample Script(6)

JMP(EB)로 강제 치환

```
return_custom_finding_binary = custom_finding_binary(regex_pattern)

# map 함수를 이용해 각 리스트의 int형 주소를 hex로 변경
finded_Obfuscation_jump_code = list((map(hex, return_custom_finding_binary)))

for foj_i in finded_Obfuscation_jump_code:
    if foj_i != idc.BADADDR: # 0xFFFFFFFF
        foj_i = int(foj_i.rstrip("L"), 0)
        # 찾은 값을 JMP로 강제 치환
        idc.PatchByte(foj_i+0x2, 235) # 235(EB == JMP)
```

ShadowPad Malware Sample

```
seg000:0000F52B ; =============== S U B R O U T I N E =======================================
seg000:0000F52B
seg000:0000F52B ; Attributes: bp-based frame
seg000:0000F52B
seg000:0000F52B sub_F52B        proc near               ; CODE XREF: sub_F51F+1↑p
seg000:0000F52B
seg000:0000F52B var_434         = byte ptr -434h
seg000:0000F52B var_34          = dword ptr -34h
seg000:0000F52B var_30          = dword ptr -30h
seg000:0000F52B var_2C          = dword ptr -2Ch
seg000:0000F52B var_28          = dword ptr -28h
seg000:0000F52B var_24          = dword ptr -24h
seg000:0000F52B var_20          = dword ptr -20h
seg000:0000F52B var_1C          = dword ptr -1Ch
seg000:0000F52B var_18          = dword ptr -18h
seg000:0000F52B var_14          = dword ptr -14h
seg000:0000F52B var_10          = dword ptr -10h
seg000:0000F52B var_C           = dword ptr -0Ch
seg000:0000F52B var_8           = dword ptr -8
seg000:0000F52B var_4           = dword ptr -4
seg000:0000F52B arg_0           = dword ptr  8
seg000:0000F52B
seg000:0000F52B                 push    ebp
seg000:0000F52C                 mov     ebp, esp
seg000:0000F52E                 sub     esp, 434h
seg000:0000F534                 push    ebx
seg000:0000F535                 push    esi
seg000:0000F536                 push    edi
seg000:0000F537                 js      short loc_F53C
seg000:0000F539                 jmp     short loc_F53C
seg000:0000F539 ; ---------------------------------------------------------------------------
seg000:0000F53B                 db 0E9h ;
seg000:0000F53C ; ---------------------------------------------------------------------------
```

스크립트 실행 결과 깔끔한 코드 확인

```
for ( i = *(_DWORD **)(*(_DWORD *)(__readfsdword(48) + 12) + 12); ; i = (_DWORD *)*i )
{
  if ( !i[6] )
    return 1;
  v2 = (_WORD *)i[12];
  v52 = 0;
  if ( *v2 )
  {
    do
    {
      v3 = __ROR4__(v52, 8);
      v4 = ((*(unsigned __int8 *)v2 | 0x20) + v3) ^ 0x7C35D9A3;
      ++v2;
      v52 = (_DWORD *)v4;
    }
    while ( *v2 );
    if ( v4 == 0xFD5B1261 )
      break;
  }
}
```

ShadowPad Malware Sample

난독화 코드를 탐지할 수 있는 yara 룰 작성

```
rule xshell_obfuscation
{
    meta:
        author = "Kang Hyeong Min"
        description = "obfuscated binaries of xshell malware"

    strings:
        $obfuscated_jmp_code = { 7? 03 7? 01 (e9|e8) }
    condition:
        // 탐지된 룰이 3개 이상일때 탐지
        #obfuscated_jmp_code > 3
}
```

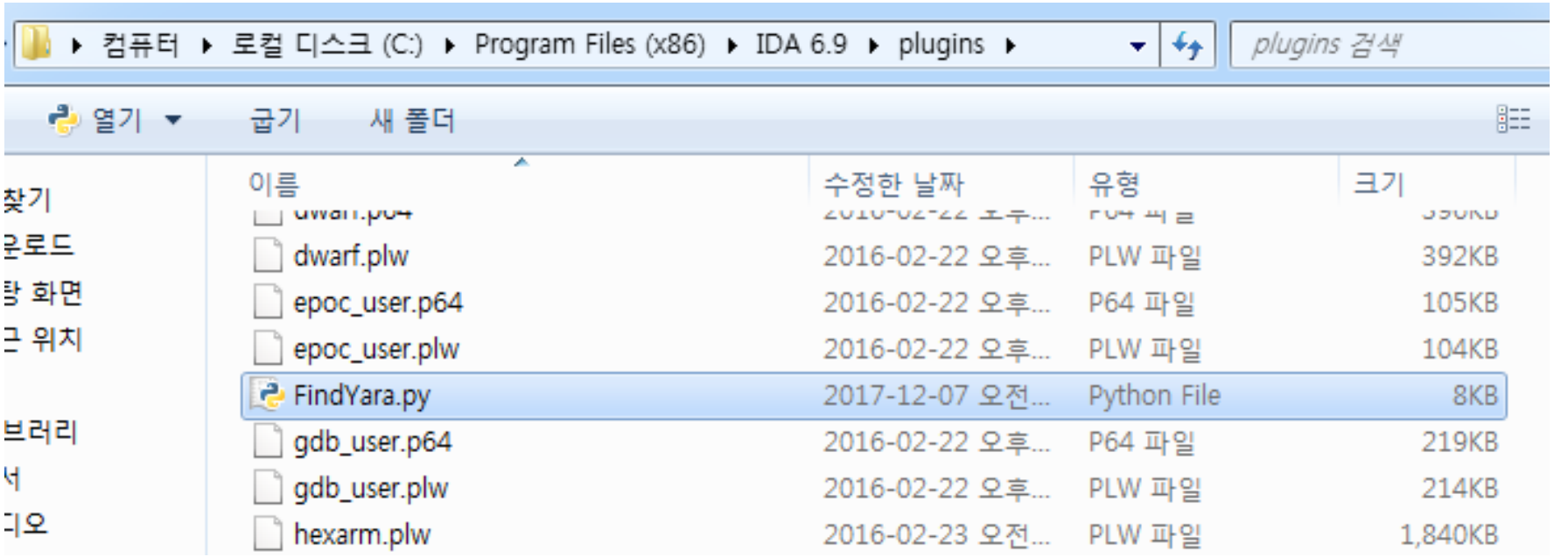
ShadowPad Malware Sample

Yara 룰 테스트 결과, 탐지되는 것을 확인

```
0xf537:$obfuscated_jmp_code: 78 03 79 01 E9
0xf557:$obfuscated_jmp_code: 78 03 79 01 E9
0xf581:$obfuscated_jmp_code: 7C 03 7D 01 E9
0xf5b3:$obfuscated_jmp_code: 70 03 71 01 E9
0xf5c0:$obfuscated_jmp_code: 7B 03 7A 01 E8
0xf5e2:$obfuscated_jmp_code: 7F 03 7E 01 E8
0xf647:$obfuscated_jmp_code: 78 03 79 01 E9
0xf674:$obfuscated_jmp_code: 73 03 72 01 E8
0xf68d:$obfuscated_jmp_code: 74 03 75 01 E9
0xf6a1:$obfuscated_jmp_code: 7C 03 7D 01 E9
0xf6ea:$obfuscated_jmp_code: 7F 03 7E 01 E8
0xf705:$obfuscated_jmp_code: 74 03 75 01 E9
0xf728:$obfuscated_jmp_code: 7B 03 7A 01 E8
0xf741:$obfuscated_jmp_code: 7C 03 7D 01 E9
0xf7a2:$obfuscated_jmp_code: 7F 03 7E 01 E8
0xf7b7:$obfuscated_jmp_code: 78 03 79 01 E9
0xf82e:$obfuscated_jmp_code: 77 03 76 01 E8
0xf840:$obfuscated_jmp_code: 7B 03 7A 01 E8
0xf851:$obfuscated_jmp_code: 7C 03 7D 01 E9
0xf86c:$obfuscated_jmp_code: 73 03 72 01 E8
0xf876:$obfuscated_jmp_code: 77 03 76 01 E8
0xf88f:$obfuscated_jmp_code: 78 03 79 01 E9
0xf89c:$obfuscated_jmp_code: 73 03 72 01 E8
0xf8c1:$obfuscated_jmp_code: 7C 03 7D 01 E9
0xf8d5:$obfuscated_jmp_code: 74 03 75 01 E9
0xf8fb:$obfuscated_jmp_code: 70 03 71 01 E9
0xf919:$obfuscated_jmp_code: 7C 03 7D 01 E9
0xf940:$obfuscated_jmp_code: 7B 03 7A 01 E8
0xf975:$obfuscated_jmp_code: 74 03 75 01 E9
0xf991:$obfuscated_jmp_code: 7C 03 7D 01 E9
0xf9a8:$obfuscated_jmp_code: 7B 03 7A 01 E8
0xf9ce:$obfuscated_jmp_code: 77 03 76 01 E8
0xf9ec:$obfuscated_jmp_code: 73 03 72 01 E8
0xfa13:$obfuscated_jmp_code: 70 03 71 01 E9
0xfa3c:$obfuscated_jmp_code: 73 03 72 01 E8
0xfa74:$obfuscated_jmp_code: 73 03 72 01 E8
0xfad8:$obfuscated_jmp_code: 7B 03 7A 01 E8
0xfaeb:$obfuscated_jmp_code: 70 03 71 01 E9
0xfaff:$obfuscated_jmp_code: 78 03 79 01 E9
0xfb2b:$obfuscated_jmp_code: 70 03 71 01 E9
```

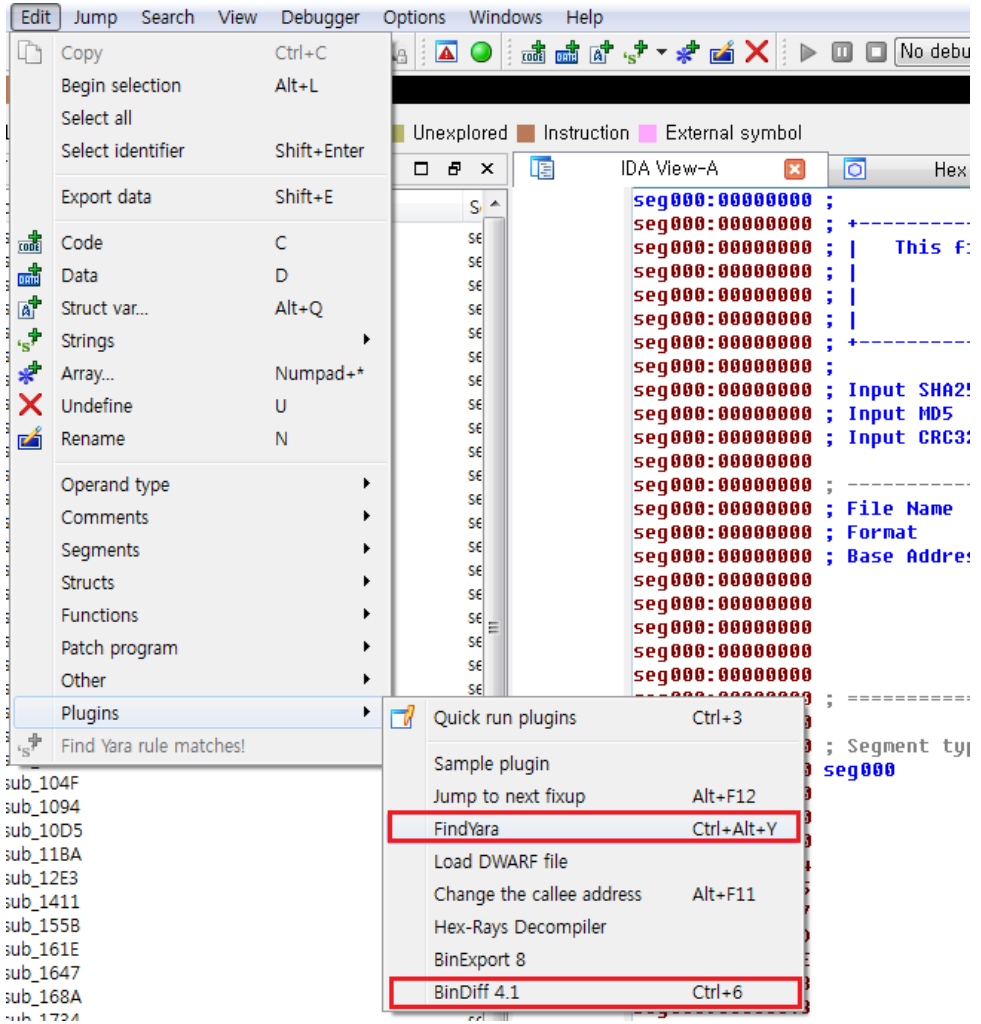
IDAPlugin FindYara

1. pip install yara-python
2. git clone https://github.com/OALabs/FindYara.git
3. 다운로드 받은 폴더 내부의 FindYara.py 파일을 %IDA_DIR%\plugins 경로에 복사



IDAPlugin FindYara

4. IDA – Edit – Plugins – FindYara 실행



IDAPlugin FindYara

HoyKey(Ctrl-Alt-y) 클릭 후 테스트하고싶은 룰을 선택하면 GUI 창에 주소, 룰 이름, Match 내용, Type 출력

```
================================================================================

 ____ __ __ __ ____  _ _ ___ ____ ___
||    || ||W || || WW  WW// // WW || WW // WW
||==  || ||WW|| ||  ))  )/  ||=|| ||_// ||=||
||    || || W|| ||_//  //   || || || WW || ||

FindYara v1.0 by @herrcore
* All credit to David Berard (@_p0ly_) for the code! *
* This is a slightly modified version of findcrypt-yara *

FindYara search shortcut key is Ctrl-Alt-y
================================================================================
```

FindYara scan results

Address	Rule Name	Match	Type
seg000:F537	xshell_obfuscation	78 03 79 01 e9	binary
seg000:F557	xshell_obfuscation	78 03 79 01 e9	binary
seg000:F581	xshell_obfuscation	7c 03 7d 01 e9	binary
seg000:F5B3	xshell_obfuscation	70 03 71 01 e9	binary
seg000:F5C0	xshell_obfuscation	7b 03 7a 01 e8	binary
seg000:F5E2	xshell_obfuscation	7f 03 7e 01 e8	binary
seg000:F647	xshell_obfuscation	78 03 79 01 e9	binary
seg000:F674	xshell_obfuscation	73 03 72 01 e8	binary
seg000:F68D	xshell_obfuscation	74 03 75 01 e9	binary
seg000:F6A1	xshell_obfuscation	7c 03 7d 01 e9	binary
seg000:F6EA	xshell_obfuscation	7f 03 7e 01 e8	binary
seg000:F705	xshell_obfuscation	74 03 75 01 e9	binary
seg000:F728	xshell_obfuscation	7b 03 7a 01 e8	binary
seg000:F741	xshell_obfuscation	7c 03 7d 01 e9	binary
seg000:F7A2	xshell_obfuscation	7f 03 7e 01 e8	binary
seg000:F7B7	xshell_obfuscation	78 03 79 01 e9	binary
seg000:F82E	xshell_obfuscation	77 03 76 01 e8	binary
seg000:F840	xshell_obfuscation	7b 03 7a 01 e8	binary
seg000:F851	xshell_obfuscation	7c 03 7d 01 e9	binary
seg000:F86C	xshell_obfuscation	73 03 72 01 e8	binary
seg000:F876	xshell_obfuscation	77 03 76 01 e8	binary
seg000:F88F	xshell_obfuscation	78 03 79 01 e9	binary
seg000:F89C	xshell_obfuscation	73 03 72 01 e8	binary

ShadowPad Malware Sample

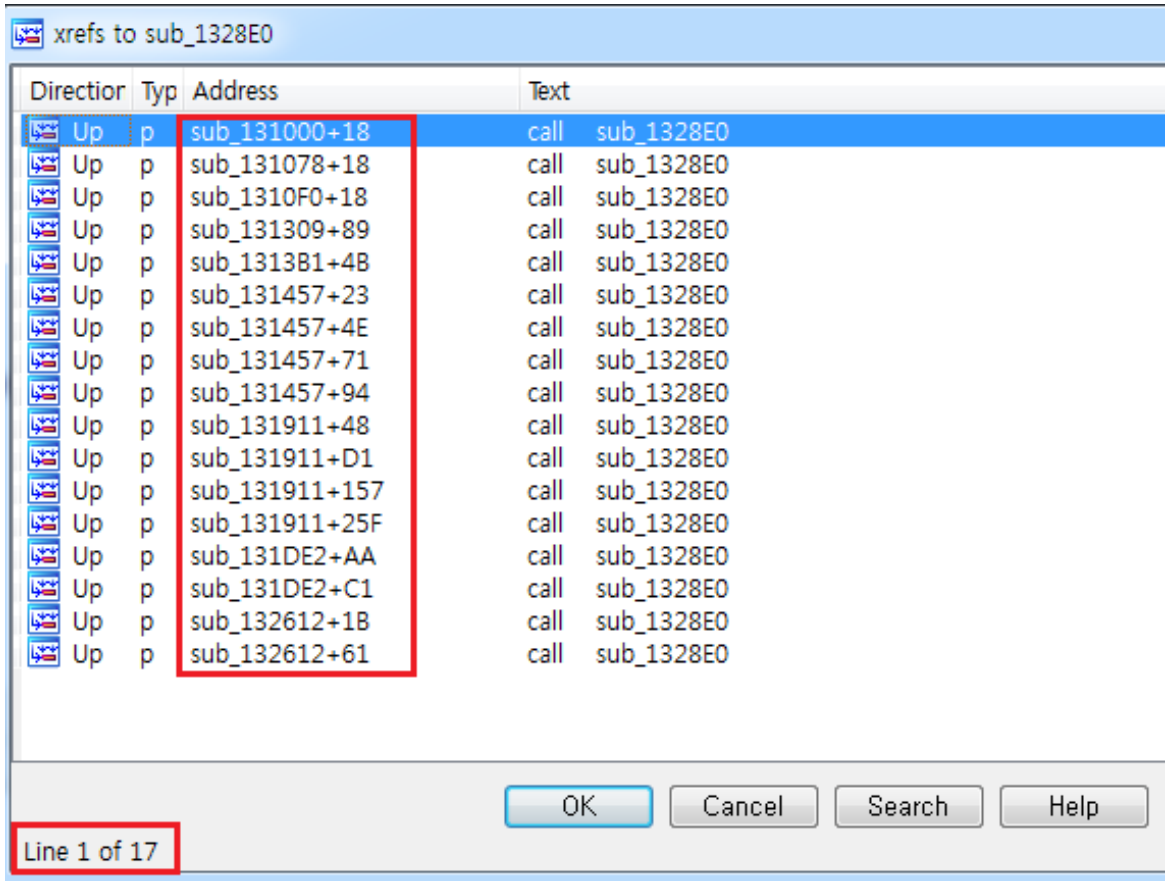
sub_1328E0 : 인코딩된 문자열 디코딩하는 함수
unk_1330C0 : 인코딩된 문자열이 존재하는 주소

```
if ( !dword_14100C )
{
  v0 = sub_1328E0((int)&unk_1330C0, (int)&v9);
  v1 = sub_13284C(v0, 0);
  dword_14100C = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))sub_131000(v1);
  sub_1328B5(v2);
}
v3 = dword_14100C(0, 0, sub_131911, 0, 0, &v10);
if ( !dword_141010 )
{
  v4 = sub_1328E0((int)&unk_1330D0, (int)&v8);
  v5 = sub_13284C(v4, 0);
  dword_141010 = (int (__stdcall *)(_DWORD))sub_131000(v5);
  sub_1328B5(v6);
}
dword_141010(v3);
return 0;
}
```

```
seg000:001330C0 unk_1330C0      db  79h ; y
seg000:001330C1                 db  17h
seg000:001330C2                 db  3Ah ; :
seg000:001330C3                 db  83h ;
seg000:001330C4                 db 0E0h ;
seg000:001330C5                 db    4
seg000:001330C6                 db 0D3h ;
seg000:001330C7                 db  7Ch ; |
seg000:001330C8                 db  81h ;
seg000:001330C9                 db  63h ; c
seg000:001330CA                 db  45h ; E
seg000:001330CB                 db  82h ;
seg000:001330CC                 db  94h ;
seg000:001330CD                 db 0D3h ;
seg000:001330CE                 db  4Bh ; K
seg000:001330CF                 db    0
```

ShadowPad Malware Sample

- sub_1328E0 함수에 대한 상호참조(xrefs to == 단축키 x) 결과, 17개 존재하는 것을 확인
- 디버깅을 통해 어떤 문자열인지 확인 가능하지만 IDA를 통한 정적분석 관점에서 스크립트 작성 필요



인코딩된 데이터를 가져와 디코딩 후 주석 및 주소 이름을 변경하는 IDAPython 스크립트 작성

XOR 디코딩(0x1328E0) 스크립트

```
_DWORD *__usercall sub_1328E0@<eax>(unsigned __int8 *a1@<eax>, _DWORD *a2@<ecx>)
{
  unsigned __int8 *v2; // edi@1
  _DWORD *v3; // esi@1
  _BYTE *v4; // ebx@1
  unsigned int v5; // eax@1
  _BYTE *v6; // ecx@1
  int v7; // edi@1
  signed int v9; // [esp+Ch] [ebp-4h]@1

  v2 = a1;
  v3 = a2;
  v9 = 0;
  v4 = (_BYTE *)sub_1325CF(4096);
  v5 = *v2 | (unsigned __int16)(v2[1] << 8);
  v6 = v4;
  v7 = v2 + 2 - v4;
  do
  {
    *v6 = v5 ^ v6[v7];
    v5 = 1091698688 * v5 - 1129103086 * (v5 >> 16) - 797412879;
    if ( !*v6 )
      break;
    ++v9;
    ++v6;
  }
  while ( v9 < 4090 );
  *v3 = 0;
  v3[1] = 0;
  v3[2] = 0;
  v3[3] = 0;
  sub_1327B2(v4);
  sub_1325A6(v4);
  return v3;
}
```

```
def decode(s):
    out = []
    cs = map(ord, s)
    v5 = cs[0] | (cs[1] << 8)
    i = 2
    while i < 4090:
        v6 = (v5 ^ cs[i]) & 0xFF
        out.append(v6)
        v5 = (0x41120000 * v5 - 0x434CBEEE * (v5 >> 16) - 0x2F878E0F) & 0xFFFFFFFF
        if v6 == 0:
            break
        i += 1
    return ''.join(map(chr, out)).rstrip('\0')
```

인코딩된 Hex 값을 XOR 디코딩 하는 스크립트

IDAPython – Sample Script(7)

- 문자열 디코딩 함수(0x1328E0)를 사용하는 상호참조 주소와 파라미터 값을 가져와 디코딩 수행
- XrefsTo(address, flags) : address를 상호참조하는 주소를 리스트 형태로 리턴
- MakeComm(address, comment) : address에 comment(주석)을 작성
- MakeName(address, text) : address의 이름을 text(변경하고자 하는 이름)로 변경

```
# XOR 디코딩 함수를 상호참조하는 주소를 for문으로 순환
for addr in XrefsTo(0x1328E0, flags=0): # 0x1328E0(XOR 디코딩 함수 주소)
    ref = find_function_arg(addr.frm) # 인코딩된 데이터 값의 실제 주소를 가져옴
    string = get_string(ref) # 인코딩된 데이터 값을 가져옴
    decoded_string = decode(string) # 인코딩된 데이터 디코드 수행
    print "Ref Addr: 0x%x | Decoded: %s" % (addr.frm, decoded_string)
    MakeComm(addr.frm, decoded_string) # 디코딩 함수 주소에 디코딩 문자열 주석
    MakeComm(ref, decoded_string) # 인코딩된 데이터 값의 실제 주소에 디코드된 문자열 주석
    remove(decoded_string, '!#^*+=|;/,-') # BAD CHARACTER 제거(오류 예외처리)
    idc.MakeName(ref, 'a_'+decoded_string.rstrip('\\')) # 디코드된 문자열로 이름 변경
```

IDAPython – Sample Script(7)

- PrevHead(address) : 이전 명령 주소(instruction address)를 리턴
- GetMnem(address) : address의 레지스터를 리턴
- GetOpnd(address, index) : address의 오퍼랜드를 리턴(index로 몇 번째 오퍼랜드 값(0부터 시작)을 가져올지 결정 가능)
- GetOperandValue(address, index) : addres의 index 오퍼랜드 값의 주소를 리턴

```
def find_function_arg(addr):
    while True:
        addr = idc.PrevHead(addr) # 해당 주소(addr)의 -1 위치의 주소를 리턴

        # GetMnem과 GetOpnd 함수를 통해 해당 주소(addr)의 레지스터, 오퍼랜드를 가져와
        # 레지스터가 mov이고 오퍼랜드의 첫 번째 인자가 eax인 것을 찾음
        # ex) mov eax, 10
        if GetMnem(addr) == "mov" and "eax" in GetOpnd(addr, 0):
            return GetOperandValue(addr, 1) # 찾은 주소(addr)의 두 번째 오퍼랜드의 실제 주소를 리턴
    return ""
```

IDAPython – Sample Script(7)

- address의 인코딩된 Byte값을 끝(0일 때까지)까지 순차적으로 가져와 문자열로 변환 후 저장
- Byte(addr) : address의 byte 값을 리턴

```
def get_string(addr):
    out = ""
    while True:
        if Byte(addr) !=0: # 인코딩된 주소의 Byte 값이 0일 때까지 반복
            out += chr(Byte(addr)) # Byte 값을 문자열로 변환 후에 out 변수에 저장
        else:
            break
        addr += 1
    return out
```

ShadowPad Malware Sample

- 디코딩된 문자열 17개(API 함수, IP 주소 등)

```
Ref Addr: 0x131018 | Decoded: kernel32.dll
Ref Addr: 0x131090 | Decoded: msvcrt.dll
Ref Addr: 0x131108 | Decoded: memcpy
Ref Addr: 0x131392 | Decoded: .com
Ref Addr: 0x1313fc | Decoded: %d
Ref Addr: 0x13147a | Decoded: 8.8.8.8
Ref Addr: 0x1314a5 | Decoded: 8.8.4.4
Ref Addr: 0x1314c8 | Decoded: 4.2.2.1
Ref Addr: 0x1314eb | Decoded: 4.2.2.2
Ref Addr: 0x131959 | Decoded: SOFTWARE\
Ref Addr: 0x1319e2 | Decoded: memset
Ref Addr: 0x131a68 | Decoded: Data
Ref Addr: 0x131b70 | Decoded: Data
Ref Addr: 0x131e8c | Decoded: GetAdaptersAddresses
Ref Addr: 0x131ea3 | Decoded: iphlpapi.dll
Ref Addr: 0x13262d | Decoded: CreateThread
Ref Addr: 0x132673 | Decoded: CloseHandle
```

ShadowPad Malware Sample

```
if ( !dword_14100C )
{
  v0 = sub_1328E0((int)&a_CreateThread, (int)&v9);
  v1 = sub_13284C(v0, 0);
  dword_14100C = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))sub_131000(v1);
  sub_1328B5(v2);
}
v3 = dword_14100C(0, 0, sub_131911, 0, 0, &v10);
if ( !dword_141010 )
{
  v4 = sub_1328E0((int)&a_CloseHandle, (int)&v8);
  v5 = sub_13284C(v4, 0);
  dword_141010 = (int (__stdcall *)(_DWORD))sub_131000(v5);
  sub_1328B5(v6);
}
dword_141010(v3);
return 0;
}
```

```
mov     eax, offset a_CreateThread
lea     ecx, [ebp+var_14]
call    sub_1328E0      ; CreateThread
push    edi
mov     esi, eax
call    sub_13284C
push    eax
call    sub_131000
pop     ecx
lea     esi, [ebp+var_14]
mov     ds:dword_14100C, eax
call    sub_1328B5
```

주소 이름 변경 및 주석 추가

```
                                        y           ; DATA XREF: sub_132612+13Îo
                                                    ; CreateThread
seg000:001330C2                 db  3Ah ; :
seg000:001330C3                 db  83h ;
seg000:001330C4                 db 0E0h ;
seg000:001330C5                 db    4
seg000:001330C6                 db 0D3h ;
seg000:001330C7                 db  7Ch ; |
seg000:001330C8                 db  81h ;
seg000:001330C9                 db  63h ; c
seg000:001330CA                 db  45h ; E
seg000:001330CB                 db  82h ;
seg000:001330CC                 db  94h ;
seg000:001330CD                 db 0D3h ;
seg000:001330CE                 db  4Bh ; K
seg000:001330CF                 db    0
```

Hex-Rays Plugin Contest

Hex-Rays에서 운영하는 Hex-Rays Plugin Contest

Hex-Rays Plugin Contest Results 2017

This year, we have examined the plugins of 7 contestants, for a grand total of 9 plugins, of wildly varying scopes & focus.

We picked three winners from those plugins; here's the final ranking:

- First prize (2000 USD): BinCAT, by Sarah Zennou, Philippe Biondi, Raphaël Rigo, Xavier Mehrenberger -- Airbus
- Second prize (1000 USD): lighthouse, by Markus Gaasedelen
- Third prize (500 USD): SimplifyGraph, by Jay Smith, FLARE Team, FireEye

Congratulations to the winners!

Below is the full list of submissions:

- ***BinCAT* by Sarah Zennou, Philippe Biondi, Raphaël Rigo, Xavier Mehrenberger -- Airbus**

  BinCAT is:

  ... a static Binary Code Analysis Toolkit, designed to help reverse engineers, directly from IDA. It features:

Hex-Rays Plugin Contest 2017 - Retdec

File Edit Jump Search View Debugger Options Windows Help

No debugger

Library function | Data | Regular function | Unexplored | Instruction | External symbol

Functions wi... | ID... | Occurren... | Pseudocod... | He... | St... | En... | Im... | Ex...

Function name

- sub_6CC80
- sub_6CD20
- sub_6CDC0
- sub_6CEB0
- sub_6CEC0
- sub_6CED0
- sub_6CF80
- sub_6D040
- sub_6D080
- sub_6D0D0
- sub_6D140
- sub_6D1B0
- sub_6D290
- sub_6D2A0
- sub_6D2B0
- sub_6D2C0
- nullsub_1
- sub_6D2E0
- nullsub_7

Line 2047 of 5959

Pseudocode-A

```
1  signed int __cdecl sub_6CED0(int a1, int a2)
2  {
3    int v2; // eax@2
4    int v3; // esi@2
5    signed int result; // eax@3
6
7    if ( !dbg )
8      return 0;
9    v2 = operator new(0x1Cu);
10   *(_DWORD *)(v2 + 4) = 0;
11   v3 = v2;
12   sub_6C9D0(v2, a2);
13   if ( *(_DWORD *)(a1 + 8) )
14   {
15     **(_DWORD **)(a1 + 12) = v3;
16     result = 1;
17     *(_DWORD *)(a1 + 12) = v3;
18   }
19   else
20   {
21     *(_DWORD *)(a1 + 12) = v3;
22     result = 1;
23     *(_DWORD *)(a1 + 8) = v3;
24   }
25   return result;
26 }
```

0006CED0 sub_6CED0:11

RetDec

```
int32_t sub_6C9D0(int32_t a1, int32_t a2);
int32_t sub_6CED0(int32_t a1, int32_t a2);

// ------------------------ Functions ------------

// Address range: 0x6ced0 - 0x6cf6c
int32_t sub_6CED0(int32_t a1, int32_t a2) {
    // entry
    sub_1D457(0);
    if (*(int32_t *)0x44c7fff0 == 0) {
        // 0x6cf48
        return 0;
    }
    int32_t v1 = __Znwj(28); // 0x6cf01
    *(int32_t *)(v1 + 4) = 0;
    sub_6C9D0(v1, a2);
    if (*(int32_t *)(a1 + 8) == 0) {
        // 0x6cf60
        *(int32_t *)(a1 + 12) = v1;
        *(int32_t *)(a1 + 8) = v1;
        // branch -> 0x6cf33
    } else {
        // 0x6cf26
        *(int32_t *)*(int32_t *)(a1 + 12) = v1;
        *(int32_t *)(a1 + 12) = v1;
        // branch -> 0x6cf33
    }
    // 0x6cf33
    return 1;
}

// --------------------- Meta-Information ---------
```

Output window

```
--color-for-ida
-o
'/home/aundro/ida-6.95-tests/idal.c'
--select-decode-only
--select-ranges='0x6ced0-0x6cf6c,0x1d457-0x1d45a,0x16130-0x16135,0x6c9d0-0x6cbe1,0x8b210-0x8b2b2'
[RetDec info]    :    Running the decompilation command ...
[RetDec info]    :    Decompiled file: /home/aundro/ida-6.95-tests/idal.c
```

Python

백신업체 어베스트가 오픈소스로 공개한 멀웨어 디컴파일러

어베스트가 멀웨어 디컴파일러를 오픈소스로 기증했다

좋아요 117개 | 입력 : 2017-12-14 15:45

멀웨어 디컴파일하는 '렛덱', 멀웨어 원리 등 파악 도와
어베스트, "오픈소스화 함으로써 렛덱 더 개발되길 기대"

[보안뉴스 오다인 기자] 백신 회사 어베스트(Avast)가 오픈소스 커뮤니티에 자체 개발한 멀웨어 디컴파일러(decompiler) 툴을 기증했다.

[이미지=iclickart]

가장 많이 본 기사

1 [보안다반사] 간호
2 KMSPico 인증 툴
3 95% 비중 가상화
4 비트코인 등 암호화
5 [단독] "유빗에는 0
6 [긴급] 사이니 멤버
7 해킹사례 전파하며
8 2018 보안 키워드
9 [연말기획] 2018년
10 [연말기획] 스마트
11 페이스북 사용자들

suspected.tistory.com

감사합니다