Checking for exported symbols/functions in a DLL without loading it

pasztorpisti, 20 Dec 2010 CPOL

4.85 (22 votes)

You can find out if a DLL exports some symbols/functions without calling LoadLibrary() on it

The tip/trick title speaks for itself. I give you a simple function that takes a file name and a list of symbols/function names, and it returns `true` if the specified file is a DLL that exports all of the specified symbols. Detecting exports by ordinal numbers are not supported because I wasn't in need of that, but if someone needs it and I have the time, then I can put it in easily because it's easier to check than the exports by names. All this is done without loading the DLL, thus avoiding execution of a possibly dangerous `DllMain()` of an unwanted DLL in your process. It works with both 32 and 64 bit binaries. I used it in a plugin system to safely detect some plugins of older versions in a program. Use this code only when reasonable because checking all DLLs with this before all `LoadLibrary()` calls can slow down your plugin initialization code in a system with lot of plugins (eg.: GIMP)!!!

That's all folks!

stdafx.h:

```
#pragma once

#include <windows.h>
#include <winnt.h> // must be after windows.h
#include <vector>
#include <set>
#include <string>
#include <cassert>
```

stdafx.cpp:

```
#include "stdafx.h"
```

CheckForDllExportedSymbols.h:

```
#ifndef __CHECK_FOR_DLL_EXPORTED_SYMBOLS_H__
#define __CHECK_FOR_DLL_EXPORTED_SYMBOLS_H__
#pragma once

enum ECFESResult
{
    eCFES_OK,
    eCFES_MissingFunctions,
```

```
    eCFES_ErrorOpeningFile,
    eCFES_ErrorReadingFile,
    eCFES_InvalidDosHeader,
    eCFES_InvalidNTHeader,
    eCFES_NotDLL,
    eCFES_DLLStructureError,
};

// Checks if the specified file is a real DLL that exports all
// the specified functions.
ECFESResult CheckForExportedSymbols(LPCTSTR dll_path,
    const char* symbols[], int symbol_count);

#endif
```

CheckForDllExportedSymbols.cpp:

```
#include "stdafx.h"
#include "CheckForDllExportedSymbols.h"

struct SRVAToFileOffset : public std::vector<IMAGE_SECTION_HEADER>
{
    DWORD operator()(DWORD RVA) const
    {
        for (const_iterator it=begin(), eit = end(); it != eit; ++it)
        {
            if (RVA >= it->VirtualAddress && RVA < it->VirtualAddress +
                  it->Misc.VirtualSize)
            return RVA - it->VirtualAddress + it->PointerToRawData;
        }

        return RVA;
    }
};

//-----------------------------------------------------------------------

class CCFESFile
{
    public:
    CCFESFile() : m_hFile(INVALID_HANDLE_VALUE) {}

    ~CCFESFile()
    {
        Close();
    }

    bool OpenForRead(LPCTSTR file_path)
```

```
    {
        Close();
        m_hFile = CreateFile(file_path, GENERIC_READ, FILE_SHARE_READ, NULL,
            OPEN_EXISTING, 0, NULL);
        return m_hFile != INVALID_HANDLE_VALUE;
    }

    void Close()
    {
        if (m_hFile != INVALID_HANDLE_VALUE)
        {
            ::CloseHandle(m_hFile);
            m_hFile = INVALID_HANDLE_VALUE;
        }
    }

    bool Read(void* buffer, DWORD bytes_to_read,
        DWORD file_offset = 0xFFFFFFFF, DWORD* bytes_read = NULL)
    {
        assert(m_hFile != INVALID_HANDLE_VALUE);

        if (m_hFile == INVALID_HANDLE_VALUE)
            return false;

        if (file_offset < 0xFFFFFFFF)
        {
            DWORD new_file_offset = SetFilePointer(m_hFile, (LONG)file_offset,
                NULL, FILE_BEGIN);

            if (new_file_offset != file_offset)
                return false;
        }

        DWORD read;

        if (!ReadFile(m_hFile, buffer, bytes_to_read, &read, NULL))
            return false;

        if (!bytes_read)
            return read == bytes_to_read;

        *bytes_read = read;

        return true;
    }

    private:
    HANDLE m_hFile;
};
```

```
//---------------------------------------------------------------------------

class CCheckForExportedSymbols
{
public:
    CCheckForExportedSymbols();
    ECFESResult CheckForExportedSymbols(LPCTSTR dll_path,
        const char* symbols[], int symbol_count);

private:
    bool ReadHeaders(int symbol_count);
    bool CheckSymbols(const char* symbols[], int symbol_count);
    bool Error(ECFESResult res);
    bool Read(void* buffer, DWORD bytes_to_read,
        DWORD file_offset = 0xFFFFFFFF);

private:
    ECFESResult m_Result;
    CCFESFile m_File;
    IMAGE_DATA_DIRECTORY m_ExportDirLocation; // RVA and size
    SRVAToFileOffset m_RVAToFileOffset;
};

CCheckForExportedSymbols::CCheckForExportedSymbols()
: m_Result(eCFES_OK)
{
    memset(&m_ExportDirLocation, 0, sizeof(m_ExportDirLocation));
}

bool CCheckForExportedSymbols::ReadHeaders(int symbol_count)
{
    IMAGE_DOS_HEADER dos_hdr;

    if (!Read(&dos_hdr, sizeof(dos_hdr)))
        return false;

    if (dos_hdr.e_magic != IMAGE_DOS_SIGNATURE)
        return Error(eCFES_InvalidDosHeader);

    IMAGE_NT_HEADERS64 hdr;

    if (!Read(&hdr, sizeof(hdr)
        - sizeof(hdr.OptionalHeader) + sizeof(WORD), dos_hdr.e_lfanew))
        return false;

    if (hdr.Signature != IMAGE_NT_SIGNATURE)
        return Error(eCFES_InvalidNTHeader);
```

```
if (hdr.OptionalHeader.Magic !=
    IMAGE_NT_OPTIONAL_HDR32_MAGIC &&
    hdr.OptionalHeader.Magic != IMAGE_NT_OPTIONAL_HDR64_MAGIC)
    return Error(eCFES_InvalidNTHeader);

if ((hdr.FileHeader.Characteristics & IMAGE_FILE_DLL) == 0)
    return Error(eCFES_NotDLL);

// In case of zero function names we only check for the validity of the DLL
if (symbol_count <= 0)
    return Error(eCFES_OK);

DWORD opt_hdr_size = min((DWORD)sizeof(hdr.OptionalHeader),
    (DWORD)hdr.FileHeader.SizeOfOptionalHeader);

if (hdr.OptionalHeader.Magic == IMAGE_NT_OPTIONAL_HDR32_MAGIC)
{
    // 32 bit binary
    if (opt_hdr_size < offsetof(IMAGE_OPTIONAL_HEADER32,
        DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT+1]))
        return Error(eCFES_MissingFunctions);

    if (!Read((WORD*)&hdr.OptionalHeader.Magic + 1,
        opt_hdr_size - sizeof(WORD)))
        return false;

    IMAGE_NT_HEADERS32 &hdr32 = *(IMAGE_NT_HEADERS32*) & hdr;

    if (hdr32.OptionalHeader.NumberOfRvaAndSizes
        < IMAGE_DIRECTORY_ENTRY_EXPORT + 1)
        return Error(eCFES_MissingFunctions);

    m_ExportDirLocation =
        hdr32.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT];
}
else
{
    // 64 bit binary
    if (opt_hdr_size < offsetof(IMAGE_OPTIONAL_HEADER64,
        DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT+1]))
        return Error(eCFES_MissingFunctions);

    if (!Read((WORD*)&hdr.OptionalHeader.Magic + 1,
        opt_hdr_size - sizeof(WORD)))
        return false;

    if (hdr.OptionalHeader.NumberOfRvaAndSizes
        < IMAGE_DIRECTORY_ENTRY_EXPORT + 1)
        return Error(eCFES_MissingFunctions);
```

```
        m_ExportDirLocation =
            hdr.OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT];
    }

    // reading the section headers in order to be able to calculate file
    // offsets from RVAs
    if (hdr.FileHeader.NumberOfSections)
    {
        m_RVAToFileOffset.resize(hdr.FileHeader.NumberOfSections);
        DWORD section_header_file_offset = dos_hdr.e_lfanew + sizeof(
            hdr.Signature) + sizeof(hdr.FileHeader)
            + hdr.FileHeader.SizeOfOptionalHeader;

        if (!Read(&m_RVAToFileOffset[0], hdr.FileHeader.NumberOfSections
            * sizeof(IMAGE_SECTION_HEADER), section_header_file_offset))
            return false;
    }

    return true;
}

bool CCheckForExportedSymbols::CheckSymbols(const char* symbols[],
    int symbol_count)
{
    // finding out the length of the longest function name that we are
    // looking for

    std::set
        <DWORD> symbol_lengths;

    DWORD max_funcname_len = 0;

    for (int i = 0; i < symbol_count; ++i)
    {
        DWORD len = (DWORD)lstrlenA(symbols[i]);
        symbol_lengths.insert(len);
        max_funcname_len = max(max_funcname_len, len);
    }

    // reading the export directory
    assert(m_ExportDirLocation.Size >= sizeof(IMAGE_EXPORT_DIRECTORY));

    IMAGE_EXPORT_DIRECTORY export_dir;

    if (!Read(&export_dir, sizeof(export_dir), m_RVAToFileOffset(
        m_ExportDirLocation.VirtualAddress)))
        return false;
```

```
if (export_dir.NumberOfNames < (DWORD)symbol_count)
    return Error(eCFES_MissingFunctions);

// Collecting the function names exported by the DLL. We omit function
// names whose length doesn't match
// the length of any of the function names that we are searching for.
std::vector<DWORD> exported_symbol_rvas;

exported_symbol_rvas.resize(export_dir.NumberOfNames);

if (!Read(&exported_symbol_rvas[0], sizeof(DWORD)*export_dir.NumberOfNames,
    m_RVAToFileOffset(export_dir.AddressOfNames)))
    return false;

std::set
    <std::string> exported_symbols;

std::vector<char> buf(max_funcname_len + 1, 0);

for (std::vector<DWORD>::const_iterator it = exported_symbol_rvas.begin(),
    eit = exported_symbol_rvas.end(); it != eit; ++it)
{
    DWORD read;

    if (!m_File.Read(&buf[0], max_funcname_len + 1, m_RVAToFileOffset(*it),
        &read))
        return Error(eCFES_ErrorReadingFile);

    DWORD buf_size = min(max_funcname_len + 1, read);

    DWORD len = 0;

    for (DWORD i = 0; i < buf_size; ++i)
    {
        if (!buf[i])
            break;

        ++len;
    }

    if (len >= buf_size)
        continue;

    if (!symbol_lengths.count(len))
        continue;

    exported_symbols.insert(&buf[0]);
}
```

```
    if ((int)exported_symbols.size() < symbol_count)
        return Error(eCFES_MissingFunctions);

    for (int i = 0; i < symbol_count; ++i)
    {
        if (!exported_symbols.count(symbols[i]))
            return Error(eCFES_MissingFunctions);
    }

    return Error(eCFES_OK);
}

bool CCheckForExportedSymbols::Error(ECFESResult res)
{
    m_Result = res;
    return false;
}

bool CCheckForExportedSymbols::Read(void* buffer, DWORD bytes_to_read,
    DWORD file_offset)
{
    DWORD read;

    if (!m_File.Read(buffer, bytes_to_read, file_offset, &read))
        return Error(eCFES_ErrorReadingFile);

    if (read != bytes_to_read)
        return Error(eCFES_DLLStructureError);

    return true;
}

ECFESResult CCheckForExportedSymbols::CheckForExportedSymbols(LPCTSTR dll_path,
    const char* symbols[], int symbol_count)
{
    if (!m_File.OpenForRead(dll_path))
        return eCFES_ErrorOpeningFile;

    if (!ReadHeaders(symbol_count))
        return m_Result;

    if (!CheckSymbols(symbols, symbol_count))
        return m_Result;

    return eCFES_OK;
}

//---------------------------------------------------------------------------
```

```
ECFESResult CheckForExportedSymbols(LPCTSTR dll_path, const char* symbols[],
    int symbol_count)
{
    CCheckForExportedSymbols CFES;
    return CFES.CheckForExportedSymbols(dll_path, symbols, symbol_count);
}
```

Test.cpp:

```
#include "stdafx.h"
#include "CheckForDllExportedSymbols.h"
#include <cstdio>
int main()
{
    static const TCHAR DLL[] = TEXT("e:\\PluginDLL.dll");
    static const char* EXPORTED_SYMBOLS[] = { "fnPluginDLL" };

    printf("%d\n", CheckForExportedSymbols(DLL,
        EXPORTED_SYMBOLS,
        sizeof(EXPORTED_SYMBOLS) / sizeof(EXPORTED_SYMBOLS[0])));

    return 0;
}
```

License

About the Author

pasztorpisti

Software Developer (Senior)

United Kingdom

No Biography provided