

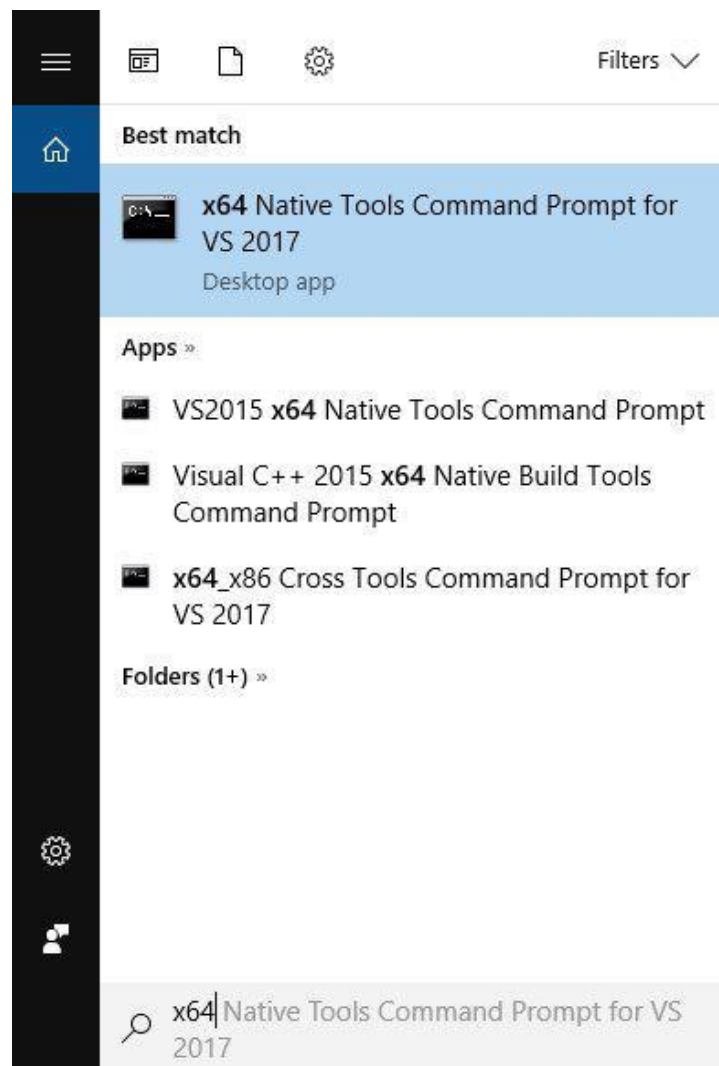
All about DLL

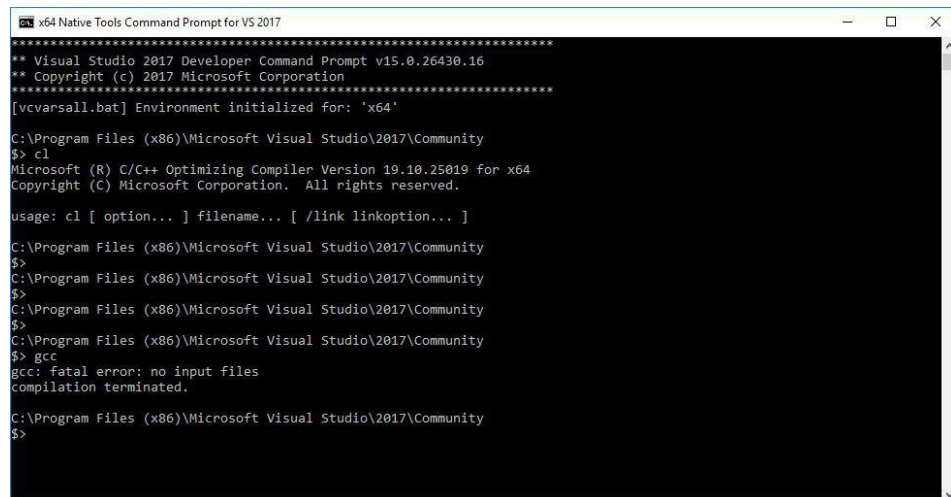
Halo selamat malam, terima kasih sudah menyempatkan datang dan main-main bersama membahas segala hal tentang DLL. Materiku akan berkisar tentang konsep DLL, cara buatnya, penggunaan, sampai preload DLL. Kita menggunakan C/C++ dengan MinGW (GCC) dan Visual Studio C++ tapi CLI compiler.

Untuk Mengikuti, silahkan nyalakan editor masing-masing dan jalankan command prompt.

Untuk menggunakan GCC pastikan sudah menginstall MinGW misal yang disediakan oleh TDM-GCC.

Untuk menggunakan Visual Studio, ada 2 opsi yaitu install Visual Studio yang Full (bersama IDE) atau install Visual Studio Build Tools. Bebas versi apapun. Untuk compile dengan Visual C++ (selanjutnya kita sebut MSVC), di start menu cari "x64 Native Tools Command Prompt for VS 2017" ato sejenisnya.





```
***** Visual Studio 2017 Developer Command Prompt v15.0.26430.16 *****
***** Copyright (c) 2017 Microsoft Corporation *****
[vcvarsall.bat] Environment initialized for: 'x64'

C:\Program Files (x86)\Microsoft Visual Studio\2017\Community
$> cl
Microsoft (R) C/C++ Optimizing Compiler Version 19.10.25019 for x64
Copyright (c) Microsoft Corporation. All rights reserved.

usage: cl [ option... ] filename... [ /link linkoption... ]

C:\Program Files (x86)\Microsoft Visual Studio\2017\Community
$>
C:\Program Files (x86)\Microsoft Visual Studio\2017\Community
$>
C:\Program Files (x86)\Microsoft Visual Studio\2017\Community
$>
C:\Program Files (x86)\Microsoft Visual Studio\2017\Community
$> gcc
gcc: fatal error: no input files
compilation terminated.

C:\Program Files (x86)\Microsoft Visual Studio\2017\Community
$>
```

Jika berhasil, kita bisa panggil compiler MSVC dengan perintah "cl", TDM-GCC bisa dipanggil dengan "gcc" seperti di gambar.

Kenapa kita pake command prompt? karena kita akan belajar mulai dari konsepnya dan memahami bagaimana barang bernama DLL ini dibuat.

Setiap source code yang akan saya berikan nanti akan ada cara kompilasinya jadi bisa direproduksi nantinya. Materi (selain log) akan saya upload ke repository github Reversing.ID

» KONSEP DLL «

Once upon a time, membuat program itu sederhana dan cenderung monolitik alias semua kode yang ada di-compile menjadi satu exe untuk dijalankan. Begitu juga ketika kita baru belajar membuat program, kita akan membuat suatu program dengan menumpahkan semua kode menjadi satu file.

Ini bagus, awalnya, tetapi masalah terjadi ketika kita ingin mengubah suatu bagian kecil, maka kita harus melakukan kompilasi ulang. Apalagi jika program itu semakin besar dan semakin kompleks. Ratusan ribu baris? Mungkin. Ubah satu atau 2 baris maka kita compile semua untuk menghasilkan satu file executable

Kemudian mulailah dilakukan modularisasi. Kode yang saling berhubungan ditulis dalam satu file yang terpisah dan akhirnya disatukan oleh yang namanya "linker". Pendekatan ini dinamakan sebagai static linking. Masalah terpecahkan. Ketika kita mau mengubah sesuatu ya lakukan untuk bagian itu saja. Misal ada bug di parsing data yang didapat dari internet, maka tinggal ubah kode ParserSuperior.cpp (misal), tak perlu mengubah source code lain.

Tapi tak semua masalah terpecahkan ... Kenapa?

Aplikasi akan tetap bloated (membengkak) untuk kode yang tak diperlukan. Karena static linking akan menghasilkan executable yang sangat besar ukurannya dengan semua kode nanti akan di-load ke dalam memori saat

program berjalan. Ini tidak efisien kala itu. Ditambah lagi, kurang bisa dilakukan reuse terhadap kode yang ada. Kode ini berguna dan telah teruji keandalannya. Bagaimana kalo kita pake di program lain? Oh bisa, masukkan saja source code-nya. Compile lagi kodenya untuk projek lain.

Muncullah ide. Bagaimana kalau kita pisahkan saja aplikasi ini menjadi pustaka-pustaka. Kode bisa kita kelompokkan jadi file tersendiri yang sudah di-compile. Aplikasi yang butuh, tinggal load ato ambil saja saat runtime. Dengan begitu kita tak perlu compile ulang kode itu berkali-kali, cukup otak-atik si program utama. Nah pustaka itu, yang diloat ketika program berjalan secara dinamis, dinamakan sebagai shared library. Di windows ekstensinya adalah .DLL, di linux/unix ekstensinya **.SO**. Proses linking ini disebut sebagai *Dynamic Linking*.

Keuntungan lainnya adalah, kita bisa outsource-kan program kita, atau gunakan komponen yang sudah ada (yang dibuat oleh orang lain).

Nah sebenarnya DLL itu makhluk macam apa? Dia sebenarnya memiliki format yang sama seperti file executable (.exe), namanya dalah PE (Portable Executable)

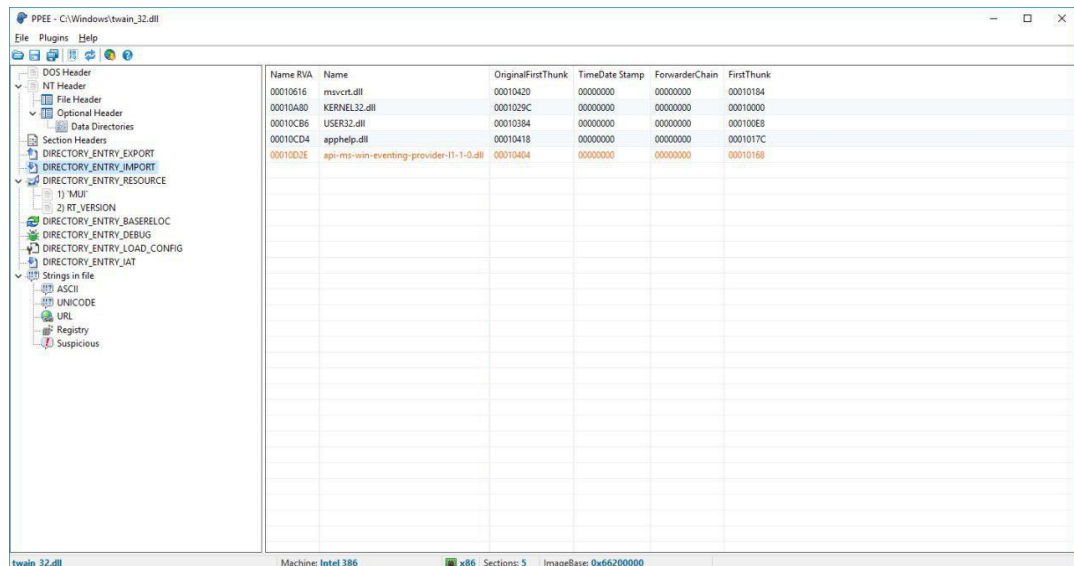
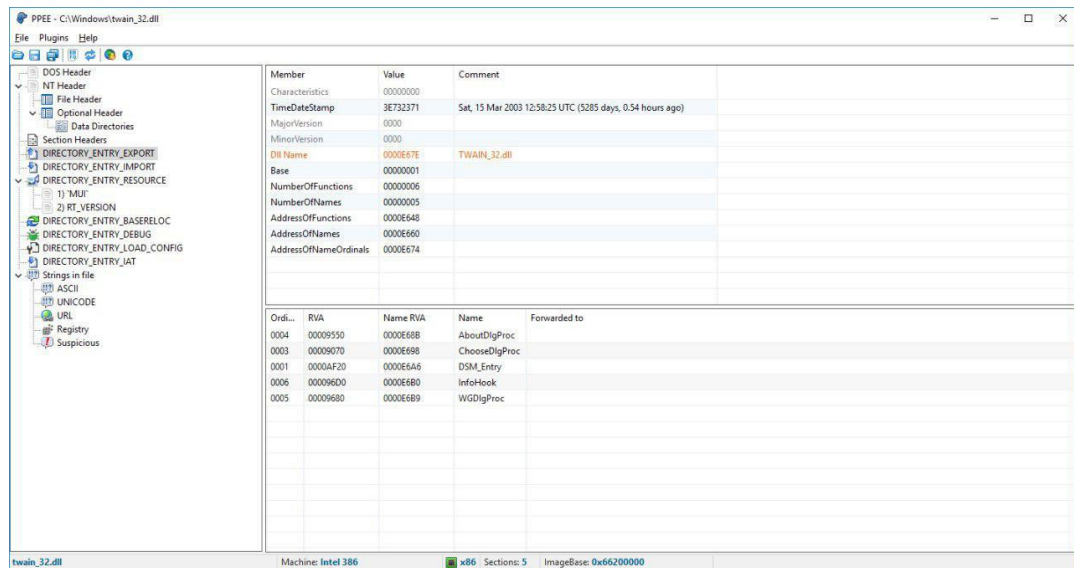
Format PE adalah format file yang mengatur tentang bagaimana kode dan instruksi program disimpan dan dijalankan nantinya. Bahasan tentang PE ini luas, tapi ada beberapa hal yang perlu kita ketahui. Dan PE ini gak hanya terbatas pada .exe aja, .dll, .scr, .cpl, .sys (windows) adalah file PE tapi beda peruntukan saja.

Nah pertanyaannya, bagaimana kode / fungsi di dalam .DLL bisa dipanggil oleh aplikasi atau DLL lain? Pertama, fungsi yang akan dipanggil itu harus dideklarasikan sebagai fungsi yang diekspor ato fungsi yang bisa dipanggil oleh pihak lain. Di sisi pemanggil, kita harus mendeklarasikan fungsi apa yang kita butuhkan.

Kenapa pemanggil harus mendeklarasikan juga?

Karena kode yang akan kita panggil itu tidak ada di kita, sehingga compiler harus diberitau bahwa fungsi ini ada di tempat lain. Dengan demikian compiler akan mengurus beberapa hal.

Silahkan buka PPEE, kemudian drag sembarang DLL yang kalian temukan di C:\Windows atau C:\Windows\System32. Ini aku ambil contoh di tempatku, Windows 10 - 64-bit, aku ambil C:\Windows\twain_32.dll



Fungsi yang kita export akan terletak di daftar namanya Export Table, sebaliknya fungsi yang kita import terletak di Import Table. Kita tidak akan bahas bagaimana strukturnya, tapi yang perlu kita tekankan adalah fungsinya. Setiap file PE punya kedua tabel itu (Export dan Import). Artinya, DLL pun bisa import fungsi dari DLL yang lain dan ia bisa export fungsi untuk dipake oleh aplikasi ato DLL lain. Aplikasi exe pun bisa export fungsi :D

Jika menggunakan contoh di foto yang kuberikan, **EXPORT** ada di **DIRECTORY_ENTRY_EXPORT**, dan fungsi yang di-export oleh DLL ini adalah AboutDlgProc, ChooseDlgProc, dsb.

Sementara fungsi yang diimport **IMPORT** ada di **DIRECTORY_ENTRY_IMPORT** tapi kita tak dapat informasi jelasnya karena kita hanya disuguhkan dengan daftar DLL yang diimport.

Dalam kasus ini ada msvcrt.dll, KERNEL32.dll, dsb.

Kalo ada yang menemukan bagian "*Forwarded to*" terisi, itu artinya pemanggilan fungsi itu akan dialihkan menuju ke DLL lain. Ini gak dibahas pada kulgram ini, dan DLL bukan hanya untuk menyimpan kode, tapi juga resources.

Untuk merangkum.

- DLL sebenarnya serupa dengan file program, tapi lebih banyak digunakan untuk menyimpan fungsi.
- Yang perlu diperhatikan adalah bagian "fungsi apa yang di-export" (EXPORT TABLE) dan "fungsi apa yang di-import" (IMPORT TABLE).

» Cara Membuat File DLL «

Intinya membuat file DLL adalah:

- Mendeklarasikan fungsi apa yang kita export.
- Menyuruh compiler untuk compile kode kita menjadi shared library.

Pendekatannya ada dua:

- memberikan penanda di source code, fungsi apa saja yang di-export.
- membuat sebuah file yang disebut Definition file, isinya menyebutkan fungsi-fungsi apa saja yang di-export.

File hello.c

```
/*
Simple DLL creation with import lib
Compilation
    using gcc
        $ gcc -shared -o hello.dll -Wl,--out-implib,libhello.a hello.c
    using MSVC
        $ cl /nologo /LD /Ox /GS /sdl hello.c
run
    rundll32 hello.dll,world
*/
#include <windows.h>

#ifdef _MSC_VER
#pragma comment(lib,"user32")
#endif

#ifdef __cplusplus
extern "C" {
#endif

void __declspec(dllexport) world()
{
    MessageBox(NULL, TEXT("This is a message"), TEXT("Title"),
        MB_OK | MB_ICONINFORMATION);
}

BOOL WINAPI _DllMainCRTStartup(HINSTANCE hinst, DWORD dwReason,
    LPVOID lpres)
{
    return 1;
}
```

```
#ifdef __cplusplus
}
#endif
```

Kita bedah dikit.

Kalo di program C, kita pasti nemu yang namanya `main()`, fungsi ini bertugas sebagai entry point alias fungsi awal yang akan dipanggil ketika program dijalankan.

DLL juga punya entry point. Jadi ketika DLL di-load ke program, dia akan menjalankan entry point ini. Gunanya adalah untuk inisialisasi, barangkali ada fungsi yang kudu dijalankan dulu sebelum fungsi lain bisa dipake.

Entry point dalam kasus ini adalah `BOOL WINAPI _DllmainCRTStartup()`. Sebenarnya ini sangat low level, ada entry point lain yang disarankan tapi karena kita bisa, ya kenapa nggak?

Kita menyatakan bahwa ada satu fungsi namanya `world()` yang di-export. Tau darimana? dari adanya `__declspec(dllexport)` sebelum nama fungsi tersebut.

Coba compile kode itu, instruksi sudah ada di file.

bagi pengguna gcc:

```
gcc -shared -o hello.dll -Wl,--out-implib,libhello.a hello.c
```

bagi pengguna MSVC:

```
cl /nologo /LD /Ox /GS /sdl hello.c
```

kemudian jalankan dengan menggunakan program `rundll32`

```
rundll32 hello.dll,world
```

apa terlihat?

```
This is a message
```

Itu tadi adalah caranya kita mendeklarasikan secara manual fungsi yang di-export di source code. Selanjutnya adalah membuat file DLL dengan cara membuat list fungsi yang di-export di definition.

File `hello2.c`

```
/*
Create DLL using definition file.
using gcc:
    $ gcc -shared -o hello2.dll -Wl,--out-implib,libhello.a hello2.c
or
    $ gcc -shared -o hello2.dll hello2.c
using MSVC:
    $ cl /nologo /LD /Ox /GS /sdl hello2.c hello2.def
run:
    rundll32 hello.dll,world
*/
#include <windows.h>

#ifdef _MSC_VER
```

```

#pragma comment(lib,"advapi32")
#pragma comment(lib,"kernel32")
#pragma comment(lib,"user32")
#endif

#ifdef __cplusplus
extern "C" {
#endif

void world()
{
    DWORD dwtemp = 62;
    TCHAR szname[64], szbuff[MAX_PATH+1];

    if (GetUserName(szname, &dwtemp))
        wprintf(szbuff, "Hello %s, how are you?", szname);
    else
        lstrcpyn(szbuff, "Hello stranger! :D", 18);
    MessageBox(NULL, szbuff, TEXT("Title"), MB_OK | MB_ICONINFORMATION);
}

BOOL WINAPI _DllMainCRTStartup(HINSTANCE hinst, DWORD dwReason,
    LPVOID lpres)
{
    return 1;
}

#ifdef __cplusplus
}
#endif

```

```

File hello2.def
LIBRARY hello2
EXPORTS
    world

```

Itu ada 2 file, hello2.c dan hello2.def

File def cukup simple, kita definisikan LIBRARY yang kita bikin namanya hello2 dan meng-EXPORTS fungsi world.

untuk GCC:

```
gcc -shared -o hello2.dll hello2.c
```

untuk MSVC:

```
cl /nologo /LD /Ox /GS /sdl hello2.c hello2.def
```

Sekali lagi, run dengan rundll32

Bagaimana dengan C++?

Ini yang agak tricky.

Untuk export yang selain kelas, kita bisa melakukan hal yang sama dengan yang kita lakukan tadi.

Lain cerita kalo kita pake kelas. Misal Semua kode implementasi kelas ada di dll, bagaimana kita export? bagaimana kita menggunakan kelasnya? bagaimana kita memanggil method di kelas itu?

Note: C++ akan melakukan proses namanya name mangling. Fitur function overload bisa terjadi karena di level assembly fungsi ini akan diberikan dekorasi yang membuat fungsi itu bernama unik, sesuai dengan tipe kembalian, tipe setiap parameter dsb sehingga fungsi yang bernama sama pun pada akhirnya namanya akan berbeda. Dan penamaan ini berbeda-beda menurut compiler.

Misal:

Sama(int a, int b) menjadi Sama_i_i

Sama(int a) menjadi Sama_i

* Hanya merupakan contoh, harap lihat pada hasil kompilasi.

File cFoo.hpp

```
class __declspec( dllexport ) CFoo
{
private:
    char m_internal[64];
public:
    CFoo();

    int add(int i, int j);
    int sub(int i, int j);
    char * last_function();
};
```

File cFoo.cpp

```
/*
DLL with exporting C++ class
Compilation
    using gcc
        $ g++ -shared -o cfoo.dll -Wl,--out-implib,libcfoo.a cfoo.cpp
    using MSVC
        $ cl /nologo /LD /Ox /GS /sdl cfoo.cpp
*/

#include <windows.h>
#include <string.h>
#include "CFoo.hpp"

#ifdef _MSC_VER
#pragma comment(lib,"user32")
#pragma warning(disable:4996)
#endif

BOOL WINAPI _DllMainCRTStartup()
{
    return 1;
}

CFoo::CFoo()
{
    memset(m_internal, 0, sizeof(m_internal));
}
```

```

int CFoo::add(int i, int j)
{
    strcpy(m_internal, "CFoo::add()");
    return (i+j);
}

int CFoo::sub(int i, int j)
{
    strcpy(m_internal, "CFoo::sub()");
    return (i-j);
}

char * CFoo::last_function()
{
    return m_internal;
}

```

Di contoh ini kita punya sebuah kelas, didefinisikan di CFoo.hpp dan diimplementasikan di CFoo.cpp, kodenya C++, dan akan kita bikin DLL-nya. Asumsiku udah pada ngerti kelas lah ya

Untuk compile di GCC:

```
g++ -shared -o cfoo.dll -Wl,--out-implib,libcfoo.a cfoo.cpp
```

Untuk compile di MSVC:

```
cl /nologo /LD /Ox /GS /sdl cfoo.cpp
```

Nah sekarang kita lihat di PPEE, liat bagian yang di-export. Gunakan kedua compiler untuk mengetahui bedanya.

Ah, sepertinya di MSVC yang baru, untuk compile harus pake command ini:

```
cl /nologo /LD /Ox /GS /sdl cfoo.cpp -D_CRT_SECURE_NO_WARNINGS
```

*-D_CRT_SECURE_NO_WARNINGS digunakan untuk men-suppress warning.

Hasil kompilasi dengan gcc

Ord...	RVA	Name RVA	Name	Forwarded to
0001	000014B0	0000806D	_Z18_DllMainCRTStartupv	
0002	00001548	00008085	_ZN4CFoo13last_functionEv	
0003	000014E6	0000809F	_ZN4CFoo3addEii	
0004	00001518	000080AF	_ZN4CFoo3subEii	
0005	000014BC	000080BF	_ZN4CFooC1Ev	
0006	000014BC	000080CC	_ZN4CFooC2Ev	

Hasil kompilasi dengan msvc

Ord...	RVA	Name RVA	Name	Forwarded to
0001	00001000	00012EFD	??0CFoo@@QEAA@XZ	
0002	000010A0	00012F0E	??4CFoo@@QEAAAEAV0\$\$QEAV0@@Z	
0003	000010D0	00012F2C	??4CFoo@@QEAAAEAV0@AEBV0@@Z	
0004	00001030	00012F48	?add@CFoo@@QEAAHHH@Z	
0005	00001080	00012F5D	?last_function@CFoo@@QEAAPEADXZ	
0006	00001050	00012F7D	?sub@CFoo@@QEAAHHH@Z	

Di situ terlihat bahwa mangling terjadi dan pembentukannya berbeda untuk compiler yang berbeda

- **File *.h mendefinisikan fungsi apa saja yang bisa dipanggil.**
- **Biasanya semua fungsi yang dipanggil itu disatukan dalam .h dan di-include**

» Menggunakan DLL «

Ada 2 cara menggunakan fungsi di DLL

- Dari awal mendefinisikan fungsi apa yang diimport.
- Di tengah jalan meload fungsi dari library.

kita fokus ke yang pertama aja ya, yang kedua silahkan ke ReversingID.

Kalo tadi kita bikin file DLL dengan mengeksport fungsi dengan `__declspec(dllexport)` maka di pemakai (exe, misalnya) kita deklarasikan sebuah function prototype dan tambahkan `__declspec(dllimport)`.

File simple.client.lib.c

```

/*
Simple client to use dll.
The dll has .lib which is used to create definition of IAT

using gcc:
$ gcc simple.client.lib.c -o simple.client.lib -L. -lhello
using MSVC:
$ cl /nologo simple.client.lib.c hello.lib

run:
    simple.client-lib.exe
*/

#include <windows.h>

#ifdef _MSC_VER
#pragma comment(lib,"kernel32")
#pragma comment(lib,"user32")
#pragma comment(linker,"/subsystem:windows /entry:main")
#endif

#ifdef __cplusplus
extern "C" {
#endif

```

```
int main()
{
    world();
    return 0;
}

#ifdef __cplusplus
}
#endif
```

Merhatiin gak pas compile hello DLL ada file extra yang dibikin? Kalau di MSVC namanya *hello.lib*, kalo di GCC namanya *libhello.a*.

Nah di windows, file lib ini dibutuhkan untuk proses "ngasih tau" kemana kita harus cari si fungsi ini tepatnya, dan untuk compile mau gak mau kita harus menyebutkan itu

GCC:

```
gcc simple.client.lib.c -o simple.client.lib.exe -L. -lhello
```

MSVC:

```
cl /nologo simple.client.lib.c hello.lib
```

jalankan:

```
simple.client.lib.exe
```

kita udah bisa panggil fungsi world() tanpa harus bikin fungsi itu.

bagaimana kalo C++? Nah ini, karena nama fungsi yang kita perlukan harus dicocokkan dengan mangling-nya, maka kita harus sesuaikan.

Sori, aku belum sempet coba yang msvc, jadi kukasih yang gcc dulu yak, tapi nanti akan di-update di github.

File class.client.cpp

```
/*
Simple client to use dll.
The dll has .lib which is used to create definition of IAT

using gcc:
$ gcc class.client.cpp -o class.client.exe -L. -lcfoo
using MSVC:
$ cl /nologo class.client.cpp cfoo.lib

run:
    simple.client.exe
*/
#include <windows.h>
#include <iostream>

#ifdef _MSC_VER
#pragma comment(lib,"kernel32")
#pragma comment(lib,"user32")
```

```

#pragma comment(linker, "/subsystem:windows /entry:main")
#endif

class __declspec( dllimport ) CFoo
{
private:
    char m_internal[64];
public:
    CFoo();

    int add(int i, int j);
    int sub(int i, int j);
    char * last_function();
};

#ifdef __cplusplus
extern "C" {
#endif

int main()
{
    CFoo cf;
    std::cout<< "135 + 135 = " << cf.add(135,135) << std::endl;

    return 0;
}

#ifdef __cplusplus
}
#endif

```

kalo sebelumnya di DLL kita deklarasikan `__declspec(dllexport)` di `CFoo.hpp`, maka kita deklarasikan kelas itu sebagai import dengan cara menggunakan `__declspec(dllimport)`

jadi, gunakan GCC:

```
g++ class.client.cpp -o class.client.exe -L. -lcfoo
```

Ingat tadi tentang name mangling? hasil DLL dari GCC akan berbeda dengan hasil DLL dari MSVC. Jika kita bikin DLL pake MSVC maka akan jadi gak compatible ketika kita compile exe dengan GCC. Jika tidak ada `libcfoo.a` maka GCC di windows akan mencari `cfoo.lib` dan masalahnya karena penamaannya berbeda maka akan terjadi error. Solusinya bagaimana? compile dengan compiler yang sama, ato jangan export class di DLL.

» PRELOAD «

Ketika aplikasi berjalan, PE loader alias bagian dari windows yang menjalankan aplikasi akan mencoba melengkapi semua fungsi dari DLL yang dibutuhkan oleh aplikasi agar bisa berjalan.

Karena DLL adalah file yang terpisah, maka tugas PE loader untuk mencari dan *me-load* DLL itu sesuai permintaan si program. pencarian DLL, PE loader akan melihat path-path berikut ini secara terurut:

- path yang sama dengan si program
- path tempat kita berada sekarang
- System directory (C:\Windows\System32)
- Windows directory (C:\Windows)
- path lain yang tertera di DLL search path

Pertanyaannya adalah, bagaimana bila ada sebuah DLL, yang namanya sama dengan DLL asli si program, dan meng-export fungsi yang sama dengan yang dibutuhkan oleh program, tapi dia terletak di direktori yang hirarkinya lebih dulu? maka DLL itu yang akan di-load terlebih dahulu.

Nah PRELOAD ini adalah kita meng-overload si DLL tadi, agar meload DLL punya kita, bukan DLL yang benar.

Lampiran:

https://github.com/ReversingID/Materi-Kulgram/tree/master/Others/IDCPLC_1_Everything%20About%20DLL