The PE File Section

الأقسام تحتوي علي المحتويات الأساسية للملف مثل الـCODE أو الـDATA أو الـResource وكل قسم له خصائصة التي يختلف بها عن القسم الاخروكما ذكرنا من قبل ذلك أن كثير من البرامج من الممكن أن تصل عدد الأقسام فيها إلي ٩ أقسام وأسمائها كالتالي : .rdata , .data , .rsrc , .edata , .idata , .pdata , .debug , .text , .bss , وبعض التطبيقات لا تطلب كل هذه الأقسام فمثلاً في مثالنا هذا يوجد ٨ أقسام وسوف يتم شرح ماهية هذه الأقسام كما يلي :-

Executable Code Section

يوجد قسم خاص يشمل Code البرنامج ويسمي هذا القسم في البرامج المبرمجة بواسطة لغة الـDelphi "**CODE**" أما في البرامج المبرمجة بلغة Microsoft Visual Basic أو Visual C++ "**.text**" ويشمل هذا القسم أيضاً نقطة دخول البرنامج وايضاً هو الذي يتحكم في سير البرنامج ويعطي البرنامج الأوامر الخاصة بعملية ما.

Data Sections

توجد عدة أقسام لبيانات البرنامج وتضم هذه الأقسام نصوص البرنامج ويسمي هذا القسم في البرامج المبرمجة بواسطة لغة الـDelphi "**DATA**" أما في البرامج المبرمجة بلغة Microsoft Visual Basic أو Visual C++ "**.bss**" و "**.data**" وقسم الـ**.rdata** إختصار read-only data وهو يشمل البيانات القابلة للقراءة فقط مثل سلاسل الحروف(النصوص) والثوابت مثل حجم الخط ودوال البرنامج.

Export data Section

القسم **.edata** يحتوي علي قائمة بالعناوين المُصدرة لملف تنفيذي أو ملف DLL.

Import data Section

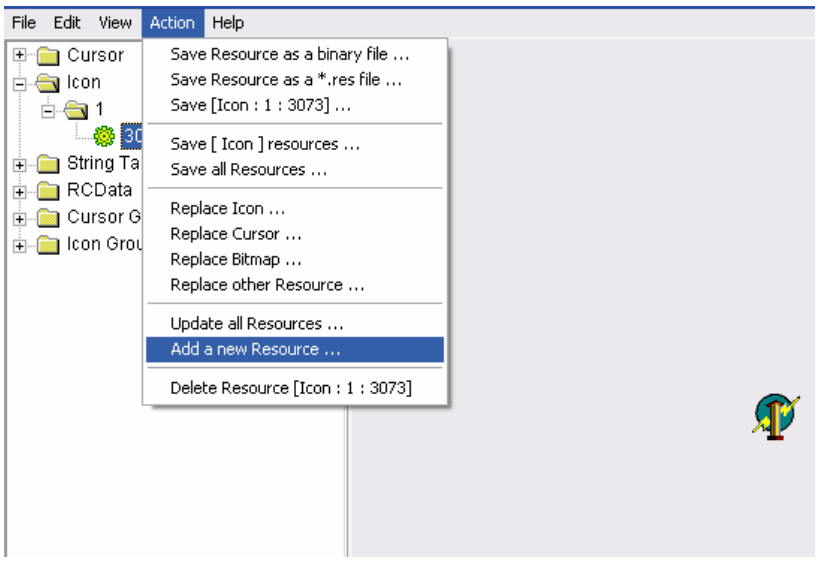
القسم **.idata** يحتوي علي قائمة بالعناوين التي تم إستيرادها داخل البرنامج سواء من ملفات الويندوز أم من ملفات أخري.

Debug information

القسم **.debug** هو الذي يشمل معلومات الـDebug الخاصة بملف الـPE وهذا القسم يتم تزويدة من قبل الـCompiler ولكن في أغلب البرامج لا يوجد هذا القسم.

Resources Section

القسم **.rsrc** يحتوي المصادر الخاصة بالبرنامج مثل الـPictures والـIcons والـDialogs الخاص بالبرنامج وبعتبر برنامج ResHacker من أفضل البرامج التي تُستخدم في تحرير البرنامج لإضافة أو إدخال أو تعديل أي شي في البرنامج من حيث الشكل العام للبرنامج ولكن ليس التعديل في الكود الخاص بالبرنامج وإنما في شكل البرنامج فقط .شغل برنامج ResHacker ثم إضغط علي Action لتري هذا الشكل :



كما تري فمن الممكن أن تُضيف Resource جديد مثل Icon أو Picture ويعتبر هذا البرنامج مفيد في حالة وجود Nag Screen لأنك تستطيع معرفتها وحذفها أو الوصول إليها في برامج الـDisAssemble عن طريق هذا البرنامج.

Base Relocation Section

عندما يتم إنشاء الملف التنفيذي بواسطة الـLinker يتم وضع إفتراض أين العنوان الذي سيبدأ به البرنامج في الـMemory وبناءاً علي ذلك يضع الـLinker العناوين الحقيقة للـCODE والـDATA داخل الملف التنفيذي. ومن الممكن أن أن تكون العناوين التي تم وضعها من قبل الـLinker تكون خاطئة ويعمل القسم **reloc.** علي السماح لـPE Loader بتصليح هذه العناوين وسوف أوضح مثال علي ذلك :-

ملف تنفيذي يبدأ بأمر مرتبط بعنوان إفتراضي 0x10000 عند الـOffset 0x2134 ولكن هذا الأمر يبدأ عند العنوان الحقيقي 0x14002 عندئذ يتم تحميل الملف ولكن الـLoader يقرر أنه من الضروري أن يبدأ البرنامج بالعنوان الحقيقي 0x60000 ، والإختلاف بين العنوان الإفتراضي والعنوان الحقيقي يسمي Delta وفي هذا المثال سوف تصبح قيمة الـDelta 0x50000 (0x10000-0x60000) عندئذ سوف يبدأ الأمر بالعنوان 0x64002 (0x14002+0x50000) ويسمي هذا إنتقال أساسي موقع الذاكره والقسم **reloc.** هو الخاص بهذه العملية ولحل هذا الإنتقال بطريقة صحيحة يقوم الـLoader بإضافة قيمة الـDelta إلي القيمة الاصلية في عنوان الإنتقال الأساسي كما حصل في هذا المثال.

The Export Table

عند يتم تحميل ملف الـPE في الـMemory بواسطة الـWindows Loader يتم تعريف هذا في الـMemory بـModule. الملف يأخذ في الـMemory شكل خريطة يتم تخطيطها وعند تخطيط الملف في الـMemory يبدأ العنوان الإفتراضي في مناداة الـHMODULE وهو يمثل الـCode والـData والـResources الذي يكون مطلوب لعملية ما من الملف التنفيذي المُحمل في الـMemory.

الـExport Table هي قائمة بدوال الـAPI أو الوظائف التي يتم تصديرها لأي ملف سواء كان هذا الملف DLL أو EXE ويتم التصدير إما بالإسم أو بواسطة العدد الترتيبي فمثلاً لو وظيفة ما تم تصديرها بواسطة الإسم ويوجد ملف DLL أو ملف تنفيذي يريد نداء هذه الوظيفة عندئذ يتم إستخدام الدالة GetProcAddress ووظيفة هذه الدالة إسترجاع الوظيفة التي تم تصديرها إلي DLL والشكل التالي يوضح إستخدام هذه الدالة :

The **GetProcAddress** function returns the address of the specified exported dynamic-link library (DLL) function.

```
FARPROC GetProcAddress(
  HMODULE hModule,        // handle to DLL module
  LPCSTR lpProcName       // name of function
);
```

Parameters

hModule
Identifies the DLL module that contains the function. The **LoadLibrary** or **GetModuleHandle** function returns this handle.

lpProcName
Points to a null-terminated string containing the function name, or specifies the function's ordinal value. If this parameter is an ordinal value, it must be in the low-order word; the high-order word must be zero.

Return Values

If the function succeeds, the return value is the address of the DLL's exported function.

If the function fails, the return value is NULL. To get extended error information, call **GetLastError**.

Remarks

The **GetProcAddress** function is used to retrieve addresses of exported functions in DLLs.

The spelling and case of the function name pointed to by *lpProcName* must be identical to that in the **EXPORTS** statement of the source DLL's module-definition (.DEF) file.

The *lpProcName* parameter can identify the DLL function by specifying an ordinal value associated with the function in the **EXPORTS** statement. **GetProcAddress** verifies that the specified ordinal is in the range 1 through the highest ordinal value exported in the .DEF file. The function then uses the ordinal as an index to read the function's address from a function table. If the .DEF file does not number the functions consecutively from 1 to *N* (where *N* is the number of exported functions), an error can occur where **GetProcAddress** returns an invalid, non-NULL address, even though there is no function with the specified ordinal.

In cases where the function may not exist, the function should be specified by name rather than by ordinal value.

كما تري فالشكل السابق يوضح فائدة هذه الدالة والإجراءات الخاصة بهذه الدالة هي

FARPROC GetProcAddress (

HMODULE hModule

LPCSTR lpProcName

):

hModule : التعرف علي الـModule الذي يحتوي علي الوظائف التي سوف يتم تصديرها للملف DLL أو EXE وتقوم الدالة LoadLibrary أو GetModuleHandle بعمل تحليل للـModule لكي يتم تحميل الوظائف أو قراءة الـModule الخاص بتصدير الوظائف.

lpProcName : يحتوي علي قيمة ترتيب الوظيفة ويوجد هنا إحتمالان :-

١- إذا كانت الدالة صحيحة عندئذ القيمة العائدة سوف تصبح العنوان الذي يحتوي علي الدالة المصدرة الصحيحة
٢- أما إذا كانت الدالة خاطئة عندئذ القيمة العائدة سوف تصبح ملغية ولبيان الخطأ يتم نداء الدالة GetLastError ومن ثم إظهار رسالة الخطأ ثم الخروج من البرنامج.

والـExport Table له بنية خاصة به مكونة من ١١ دالة كما في الشكل التالي :-

```
IMAGE_EXPORT_DIRECTORY STRUCT
  Characteristics          DWORD     ?
  TimeDateStamp            DWORD     ?
  MajorVersion             WORD      ?
  MinorVersion             WORD      ?
  nName                    DWORD     ?
  nBase                    DWORD     ?
  NumberOfFunctions        DWORD     ?
  NumberOfNames            DWORD     ?
  AddressOfFunctions       DWORD     ?
  AddressOfNames           DWORD     ?
  AddressOfNameOrdinals    DWORD     ?
IMAGE_EXPORT_DIRECTORY ENDS
```

nName : تشير هذه الدالة إلي الإسم الحقيقي للـModule مثل kernel32.dll,User32.dll ولكن هنا يكون الإسم عبارة عن إسم الملف الذي سوف يصدر له الوظائف وطبعاً هذا الـModule يشمل دوال الـAPI أو الوظائف التي سوف يتم تصديرها.

nBase : تحدد هذه الدالة العدد الترتيبي البادئ للتصدير وعادة تكون قيمة هذه الدالة تساوي "١".

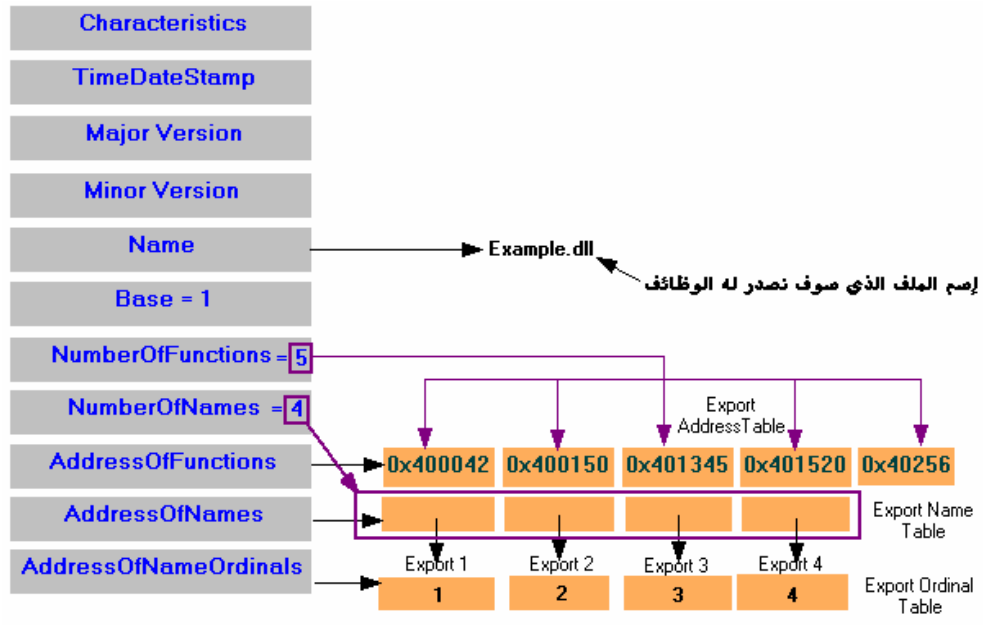
NumberOfFunctions : عدد الوظائف التي يتم تصديرها بواسطة الـModule.

NumberOfNames : عدد الوظائف التي يتم تصديرها بواسطة الإسم وهذا العدد لا يشمل كل الوظائف التي توجد في الـModule.

AddressOfFunctions : تشير هذه الدالة إلي الـRVA الخاص بالوظائف التي توجد في الـModule.

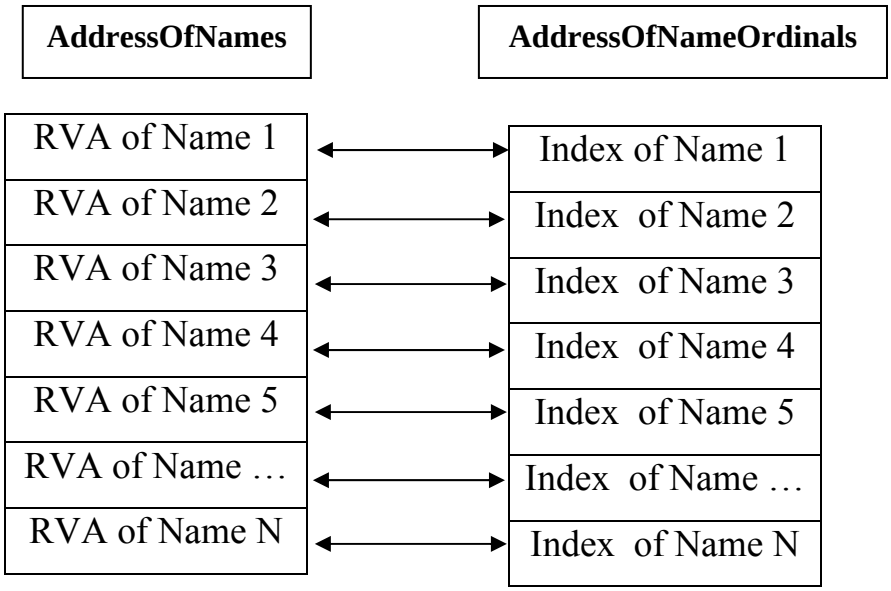
AdressOfNames : تشير هذه الدالة إلي الـRVA الخاص بأسماء الوظائف التي توجد في الـModule.

AddressOfNameOrdinals : هذه الدالة تشير إلي ترتيب الوظائف الموجودة في الدالة AddressOfNames ولكن بترتيب معين أي أن الوظائف الموجودة في الدالة AddressOfNames تكون عباره عن أسماء الوظائف ولكن هذه الدالة تأخذ الوظائف وتُرتبها والشكل التالي يوضح مفهوم Export Table كما يلي :



كما تري في الشكل السابق فملف الـPE يجب أن يجد مكان ليحفظ فيه عناوين الوظائف التي سوف يتم تصديرها لكي يأتي الـWindows Loader ليجدهم والمكان المختص بحفظ هذه العناوين هي الدالة AddressOfFunctions وعدد الوظائف يكون موجود في الدالة NumberOfFunctions فمثلاً إذا كان عدد الوظائف أو الدوال التي سوف يتم تصديرها ٥٠ فسوف تحفظهم الدالة AddressOfFunctions وعددهم سوف تشير إليهم الدالة NumberOfFunctions وسوف يحتوي القيمة ٥٠ وأيضاً إذا تم تصدير الدوال أو الوظائف عن طريق الإسم فسوف يكون إسم هذه الدوال مخزن في الدالة AddressOfNames وعدد أسماء هذه الوظائف سوف

NumberOfNames. الـLoader Windows ممكن أن يجد أسماء الوظائف في الدالة الخاصة بها وعناوين الوظائف في الدالة الخاصة بها أيضاً ولكن لا يوجد ربط بينهم لذلك يتم وضع فهارس من قبل الـPE لكي يتم الربط بينهم وهذه الفهارس يُشار إليها بالدالة AddressOfNameOrdinals أي أن هذه الدالة تستخدم لعمل فهرس لأسماء الوظائف التي يُشار إليها بالدالة AddressOfNames كما يلي :



الشكل السابق يوضح الترابط بين عناوين أسماء الوظائف وفهرسة الاسماء وكل إسم له عنوان فقط ولكن ليس العكس صحيح أي أن العنوان يمكن أن يكون مرتبط بعدة أسماء وخلاصة الموضوع أن الدالة AddressOfNameOrdinals تستخدم لعمل فهرس لأسماء الوظائف التي سوف يتم تصديرها للبرنامج وبذلك نكون قد إنتهينا من شرح الجزء الخاص بالـExport Table.

The Import Table

عندما يتم إستخدام وظائف من ملف أخر مثل DLL يسمي هذا بإستيراد وظائف لكي يتم إستخدامها من قبل الملف التنفيذي والمسئول عن هذا هو الـWindows Loder فهو الذي يعمل علي تحميل الـDLL التي سوف يتم إستيراد الوظائف منها للملف التنفيذي ومن أمثلة ملفات الـDLL التي يتم إستيراد الوظائف منها , Kernel32.dll User32.dll , Gdi32.dll , Advapi32.dll والـImport Table لها بنية خاصة بها تتكون من ٦ دوال وتحتوي علي معلومات عن ملف الـDLL التي سوف نستورد الوظائف منها والشكل التالي يوضح البنية الخاصة بالـImport Table كما يلي :

```
IMAGE_IMPORT_DESCRIPTOR STRUCT
   union
      Characteristics dd      ?
      OriginalFirstThunk dd   ?
            ends
   TimeDateStamp dd    ?
   ForwarderChain dd   ?
   Name1 dd            ?
   FirstThunk dd       ?
IMAGE_IMPORT_DESCRIPTOR ENDS
```

الدوال السابقة تمثل كل دلة منها 20 بايت ويتم توضيحها كالأتي :-

Characteristics **:** تشير إلي خصائص الـImport Table.

OriginalFirstThunk **:** تشير هذه الدالة إلي إسم الوظيفة الأولي التي سوف يتم إستيرادها.

TimeDateStamp و ForwarderChain غير مهمين بالنسبة لنا.

Name1 : تحتوي علي إسم ملف الـDLL الذي سوف نستورد منه الوظائف.

FirstThunk : تشير هذه الدالة إلي عنوان الوظيفة التي سوف يتم إستيرادها وهي لها بنية خاصة بها كما في الشكل التالي :

```
IMAGE_THUNK_DATA32 STRUCT
    union u1
        ForwarderString dd  ?
        Function dd         ?
        Ordinal dd          ?
        AddressOfData dd    ?
    ends
IMAGE_THUNK_DATA32 ENDS
```

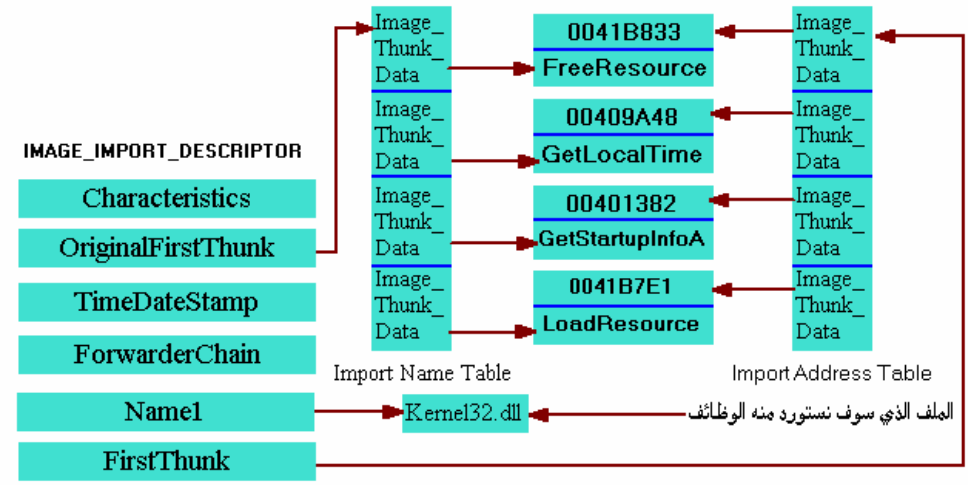
والوظائف التي يتم إستيرادها للملف التنفيذي تكون مرتبة العدد حسب الإسم وهذه أيضاً لها بنية خاصة بها كما في الشكل التالي :

```
IMAGE_IMPORT_BY_NAME STRUCT
    Hint dw     ?
    Name1 db    ?
IMAGE_IMPORT_BY_NAME ENDS
```

Hint : تحتوي هذه الدالة علي فهرس أو دليل داخل الـExport Table في ملف الـDLL الذي سوف يُصدّر الوظائف لنا أي التي نستوردها وعادة تكون قيمة هذه الدالة صفر.

Name1 : يحتوي علي إسم الوظائف المستوردة من ملف الـDLL.

والشكل التالي يوضح الـImport Table كما يلي :



كما تري فالشكل السابق يوضح إسم الملف الذي سوف نستورد منه الوظائف وأسماء الوظائف وعناوين الوظائف ويتم نداء هذه الوظائف بطريقتين فمثلاً الوظيفة GetLocalTime يتم نداءة بطريقتين :-

الطريقة الأولي :

00409A48 CALL DWORD PTR [45C2A4]

الطريقة الثانية :

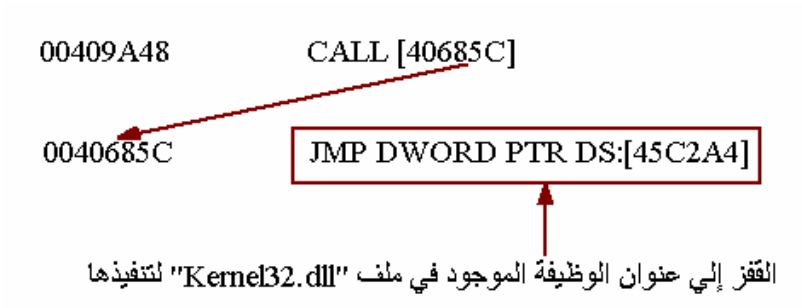
00409A48 CALL [40685C]

0040685C JMP DWORD PTR DS:[45C2A4]

كما تري فالطريقة الأولي أقصر من الطريقة الثانية وتؤدي نفس الغرض أما الطريقة الثانية تستخدم ٥ بايتات إضافية ولكن يتم إستخدام الطريقة الثانية لأن الـCompiler لا يستطيع التمييز بين النداء إلي الوظيفة العادية ضمن الملف التنفيذي والنداء إلي الوظيفة المستوردة من ملف أخر مثل "Kernel32.dll" لأن الأمر CALL [xxxxxx] يستلزم نداء مكان أو وظيفة موجودة ضمن الملف التنفيذي وليست موجودة في ملف أخر وهذا ما تمثلة الطريقة الأولي أما الطريقة الثانية فتنادي عنوان

Kernel32.dll Module لكي يتم

نداء الوظيفة المطلوبة منه والطريقة الثانية سوف يتم توضيحها في الشكل التالي :



الـImport Table يبدأ من القسم idata. والشكل التالي يوضح الـRawOffset الخاص بالبرنامج الذي نُجري علية إختبارتنا كما يلي :

[Section Table]

Name	VOffset	VSize	ROffset	RSize	Flags
CODE	00001000	00057FCC	00000400	00058000	60000020
DATA	00059000	0000111C	00058400	00001200	C0000040
BSS	0005B000	00000C51	00059600	00000000	C0000000
.idata	0005C000	000021BC	00059600	00002200	C0000040
.tls	0005F000	00000010	0005B800	00000000	C0000000
.rdata	00060000	00000018	0005B800	00000200	50000040
.reloc	00061000	00006108	0005BA00	00006200	50000040
.rsrc	00068000	00003A00	00061C00	00003A00	50000040

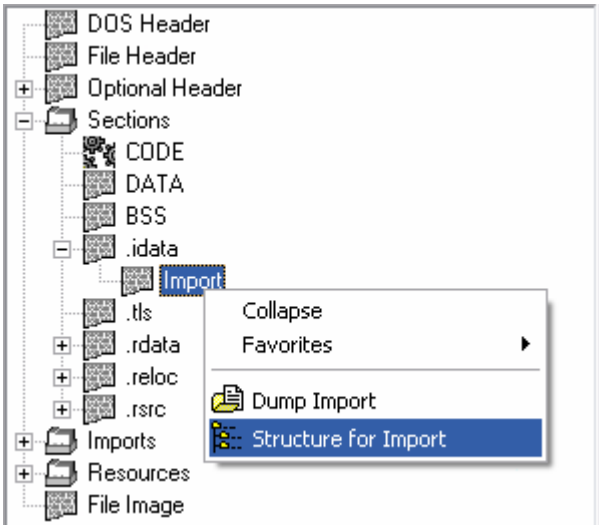
والإختلاف بين VOffset و ROffset هو 2A00 = 5C000-59600 وسوف نعرف فائدة هذه القيمة فيما بعد والأن إذا ذهبنا إلي العنوان المشار إليه في الشكل السابق في برنامج Hex Workshop سوف تري الأتي :

```
000595E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .......
000595F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .......
00059600 00 00 00 00 00 00 00 00 00 00 00 00 40 C7 05 00 .......
00059610 04 C1 05 00 00 00 00 00 00 00 00 00 00 00 00 00 .......
00059620 20 CA 05 00 AC C1 05 00 00 00 00 00 00 00 00 00  ......
00059630 00 00 00 00 66 CA 05 00 C0 C1 05 00 00 00 00 00 ....f..
00059640 00 00 00 00 00 00 00 00 A6 CA 05 00 D0 C1 05 00 .......
00059650 00 00 00 00 00 00 00 00 00 00 00 00 EE CA 05 00 .......
00059660 E0 C1 05 00 00 00 00 00 00 00 00 00 00 00 00 00 .......
00059670 3A CB 05 00 F4 C1 05 00 00 00 00 00 00 00 00 00 :......
00059680 00 00 00 00 7A CB 05 00 04 C2 05 00 00 00 00 00 ....z..
00059690 00 00 00 00 00 00 00 00 64 CF 05 00 F4 C2 05 00 .......
000596A0 00 00 00 00 00 00 00 00 00 00 00 00 E0 D3 05 00 .......
000596B0 FC C3 05 00 00 00 00 00 00 00 00 00 00 00 00 00 .......
000596C0 1E DE 05 00 80 C6 05 00 00 00 00 00 00 00 00 00 .......
000596D0 00 00 00 00 34 DE 05 00 88 C6 05 00 00 00 00 00 ....4..
000596E0 00 00 00 00 00 00 00 00 C4 DF 05 00 E4 C6 05 00 .......
000596F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .......
00059700 00 00 00 00 4E C7 05 00 66 C7 05 00 7E C7 05 00 ....N..
```

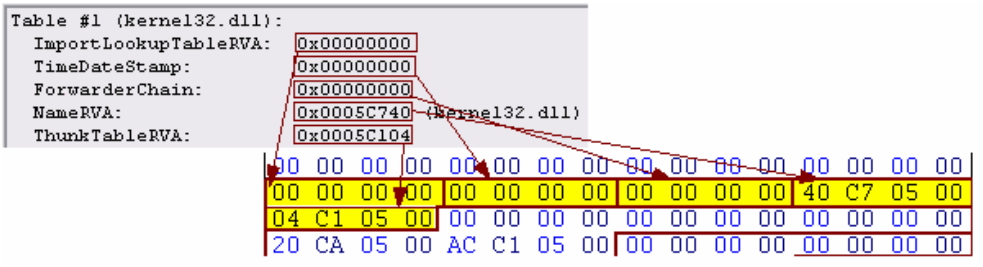
كما ذكرنا سابقاً أن الـImport Table لـه بنية خاصة بها وتتكون من ٥ دوال وهذه البنية هي IMAGE_IMPORT_DESCRIPTOR والخمس دوال هي :

OriginalFirstThunk, TimeDateStamp, ForwarderChain, Name1, FirstThunk

وإذا لاحظت أن الشكل السابق يحتوي علي ١٣ جدول وكل جدول من هذة الجداول بها تعريف للخمس دوال التي ذكرناها فإذا قمنا بتشغيل برنامج PEBrowsePro وذهبت إلي جزء الأقسام ثم إختر القسم idata. ثم إختر import ثم إضغط Right Click ثم إختر كما في الشكل التالي :



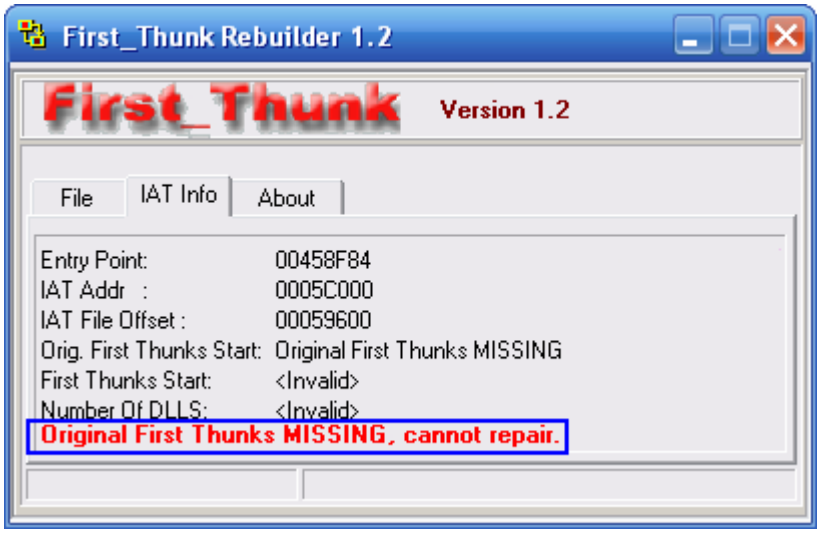
عندئذ سوف تري في الجهة اليمني ١٣ جدول وكل جدول يضم عدد من الوظائف اللازمة فمثلاً لو قارنا الجدول الأول بما يوجد لدينا في برنامج Hex Workshop سوف تري هذا الشكل :



فكما تري فالشكل السابق يوضح مطابقة الدوال الخمسة مع الجدول الأول وإذا قارنت قيم الدوال الموجودة في برنامج Hex Workshop مع قيم الجدوال الناتجة من برنامج PEBrowsePro سوف تجد أن القيم متطابقة مع العلم أن عدد الملفات التي سوف نستورد منها الوظائف ٦ ملفات وهي كما يلي :

ADVAPI32.DLL, COMCTL32.DLL, GDI32.DLL, KERNEL32.DLL, OLEAUT32.DLL, USER32.DLL

ولكن هذه الملفات تشغل ١٢ جدول والجدول الاخير قيمتة أصفار وغير مهم بالنسبة لنا ، وكما رأينا فإن قيمة أول دالة وهي OriginalFirstThunk صفر ويظهر ذلك عن طريق برنامج First_Thunk Rebuilder كما في الشكل التالي :



كمـا تـري فالـشكل الـسابق يوضـح أن الــOriginalFirstThunk قيمتهـا مفقـودة وتـساوي صـفر وهـذه القيمـة عـادة تكـون صـفر فـي البـرامج المبرمجـة بواسـطة الـBorland Delphi وأيضاً تكون صفر في البرامج المضغوطة أو المشفرة ولكن أحياناً يكون هناك نسخة منها موجودة في البرنـامج المضغوط أو المُشفر وعندما يتم فك الضغط أو التشفير يتم تصحيح هذه القيمة وإرجاعها إلي أصلها.

والدالـة Name1 فـي الجـدول الأول قيمتهـا 005C740 وسـوف نقـوم بتحويـل هـذه القيمة إلي ROffset عن طريق طرح هذه القيمة من القيمـة التـي حصلنا عليها سابقاً مـن طـرح الــVOffset مـن الــROffset وهـي 2A00 سـوف تجـد النـاتج يـساوي 005C740-2A00 = **0059D40** وإذا ذهبنـا إلـي هذا العنوان في برنـامج Hex Workshop سوف تجد الأتي :

```
00059D40 6B 65 72 6E 65 6C 33 32 2E 64 6C 6C 00 00 00 00 kernel32.dll....
00059D50 44 65 6C 65 74 65 43 72 69 74 69 63 61 6C 53 65 DeleteCriticalSe
00059D60 63 74 69 6F 6E 00 00 00 4C 65 61 76 65 43 72 69 ction...LeaveCri
00059D70 74 69 63 61 6C 53 65 63 74 69 6F 6E 00 00 00 00 ticalSection....
00059D80 45 6E 74 65 72 43 72 69 74 69 63 61 6C 53 65 63 EnterCriticalSec
00059D90 74 69 6F 6E 00 00 00 00 49 6E 69 74 69 61 6C 69 tion....Initiali
00059DA0 7A 65 43 72 69 74 69 63 61 6C 53 65 63 74 69 6F zeCriticalSectio
00059DB0 6E 00 00 00 56 69 72 74 75 61 6C 46 72 65 65 00 n...VirtualFree.
00059DC0 00 00 56 69 72 74 75 61 6C 41 6C 6C 6F 63 00 00 ..VirtualAlloc..
00059DD0 00 00 4C 6F 63 61 6C 46 72 65 65 00 00 00 4C 6F ..LocalFree...Lo
```

كما تري ففي الشكل السابق تم توضيح الدالة المستخدمة لإستيراد الوظائف المطلوبة منها أما قيمة الدالة FirstThunk الموجودة في الجدول الأول وهي 0005C104 إذا قمنا بتحويلها إلي ROffset سوف يصبح الناتج 0005C104-2A00=00059704 وإذا ذهبنا إلي هذا العنوان في برنامج Hex Workshop سوف نري هذا الشكل :

```
000596F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00059700 00 00 00 00 4E C7 05 00 66 C7 05 00 7E C7 05 00 ....N...f...~...
00059710 96 C7 05 00 B2 C7 05 00 C0 C7 05 00 D0 C7 05 00 ................
00059720 DC C7 05 00 EA C7 05 00 00 C8 05 00 18 C8 05 00 ................
00059730 30 C8 05 00 40 C8 05 00 56 C8 05 00 6C C8 05 00 0...@...V...l...
00059740 78 C8 05 00 84 C8 05 00 96 C8 05 00 A8 C8 05 00 x...............
```

وطبعاً إذا عكسنا هذه القيمة سوف تصبح 0005C74E وإذا قمنا بتحويلها إلي ROffset سوف يصبح الناتج 0005C74E-2A00 = 00059D4E وإذا ذهبنا إلي هذه العنوان سوف تري هذا الشكل :

```
00059D40 6B 65 72 6E 65 6C 33 32 2E 64 6C 6C 00 00 00 00 kernel32.dll....
00059D50 44 65 6C 65 74 65 43 72 69 74 69 63 61 6C 53 65 DeleteCriticalSe
00059D60 63 74 69 6F 6E 00 00 00 4C 65 61 76 65 43 72 69 ction...LeaveCri
```

كما تري فالشكل السابق يوضح أول وظيفة التي تم إستيرادها من الملف "Kernel32.dll" ويمكن التأكد بأن هذه أول دالة تم إستيراها من خلال برنامج PEBrowsePro كما في الشكل التالي :

```
Table #1 (kernel32.dll):
   ImportLookupTableRVA:   0x00000000
   TimeDateStamp:          0x00000000
   ForwarderChain:         0x00000000
   NameRVA:                0x0005C740 (kernel32.dll)
   ThunkTableRVA:          0x0005C104
     Thunk01 = 0x0005C74E (0, DeleteCriticalSection)
     Thunk02 = 0x0005C766 (0, LeaveCriticalSection)
```

وإذا قمنا بتحميل برنامجنا في برنامج Ollydbg ثم إضغط CTRL+N سوف تري هذا الشكل :

0045C104	.idata	Import	kernel32.DeleteCriticalSection
0045C2D8	.idata	Import	kernel32.DeleteCriticalSection
0045C3C0	.idata	Import	gdi32.DeleteDC
0045C3BC	.idata	Import	gdi32.DeleteEnhMetaFile
0045C62C	.idata	Import	user32.DeleteMenu
0045C3B8	.idata	Import	gdi32.DeleteObject

كما تري فالشكل السابق يوضح الوظائف المستوردة والوظيفة المشار إليها في الشكل السابق هي FirstThunk وإذا تذكرت قبل ذلك أن قيمة هذه الدالة في برنامج Hex Workshop هي 0005C104 وهذا هو الـRVA وإذا جمعنا الـRVA مع الـImageBase سوف يكون العنوان الناتج هو 0045C104 ألا وهو عنوان الدالة الأولي الموجود في ملف "Kernel32.dll" والمشار إليها أيضاً في الشكل السابق،إضغط عليها Right Click ثم إختر هذا الإختيار :

Address	Section	Type	Name	Comment
0045C3E4	.idata	Import	gdi32.CreateCompatibleBitmap	
0045C3E0	.idata	Import	gdi32.CreateCor	
0045C3D8	.idata	Import	gdi32.CreateDIB	
0045C3DC	.idata	Import	gdi32.CreateDIB	
0045C2E4	.idata	Import	kernel32.Create	
0045C1A0	.idata	Import	kernel32.Create	
0045C2E0	.idata	Import	kernel32.Create	
0045C3D4	.idata	Import	gdi32.CreateFor	
0045C3D0	.idata	Import	gdi32.CreateHal	
0045C648	.idata	Import	user32.CreateIc	
0045C644	.idata	Import	user32.CreateMe	
0045C3CC	.idata	Import	gdi32.CreatePal	
0045C3C8	.idata	Import	gdi32.CreatePen	
0045C640	.idata	Import	user32.CreatePo	
0045C3C4	.idata	Import	gdi32.CreateSol	
0045C2DC	.idata	Import	kernel32.Create	
0045C63C	.idata	Import	user32.CreateW	
0045C638	.idata	Import	user32.DefFrame	
0045C634	.idata	Import	user32.DefMDICh	
0045C630	.idata	Import	user32.DefWindo	
0045C104	.idata	Import	kernel32.DeleteCriticalSection	
0045C2D8	.idata	Import	kernel32.DeleteCriticalSection	

Actualize
Follow import in Disassembler
Follow in Dump
Find references to import — Enter
View call tree
Toggle breakpoint on import
Conditional breakpoint on import
Conditional log breakpoint on import
Set breakpoint on every reference
Set log breakpoint on every reference
Remove all breakpoints
Copy to clipboard
Sort by
Appearance

بعد الضغط سوف تذهب إلي هذا الشكل :

Address	Disassembly	Comment
004013D4	JMP DWORD PTR DS:[<&kernel32.DeleteCrit	ntdll.RtlDeleteCriticalSection
00401BFC	CALL <JMP.&kernel32.DeleteCriticalSecti	

الشكل السابق يوضح الأوامر الخاصة بنداء هذه الوظيفة أما إذا ضغطت Right Click ثم إخترت هذا الإختيار :

Address	Section	Type	Name	Commen
0045C3E4	.idata	Import	gdi32.CreateCompatibleBitmap	
0045C3E0	.idata	Import	gdi32.Crea	
0045C3D8	.idata	Import	gdi32.Crea	
0045C3DC	.idata	Import	gdi32.Crea	
0045C2E4	.idata	Import	kernel32.C	
0045C1A0	.idata	Import	kernel32.C	
0045C2E0	.idata	Import	kernel32.C	
0045C3D4	.idata	Import	gdi32.Crea	
0045C3D0	.idata	Import	gdi32.Crea	
0045C648	.idata	Import	user32.Cre	
0045C644	.idata	Import	user32.Cre	
0045C3CC	.idata	Import	gdi32.Crea	
0045C3C8	.idata	Import	gdi32.Crea	
0045C640	.idata	Import	user32.Cre	
0045C3C4	.idata	Import	gdi32.Crea	
0045C2DC	.idata	Import	kernel32.C	
0045C63C	.idata	Import	user32.Cre	
0045C638	.idata	Import	user32.Def	
0045C634	.idata	Import	user32.Def	
0045C630	.idata	Import	user32.Def	
0045C104	.idata	Import	kernel32.DeleteCriticalSection	

Actualize
Follow import in Disassembler
Follow in Dump
Find references to import Enter
View call tree
Toggle breakpoint on import
Conditional breakpoint on import
Conditional log breakpoint on import
Set breakpoint on every reference
Set log breakpoint on every reference
Remove all breakpoints
Copy to clipboard
Sort by
Appearance

بعد الضغط سوف تذهب إلي الأمر مباشرة كما في الشكل التالي :

Address	Hex dump	Disassembly
7C91188A	6A 1C	PUSH 1C
7C91188C	68 0819917C	PUSH ntdll.7C911908
7C911891	E8 2CD5FFFF	CALL ntdll.7C90EDC2
7C911896	8B5D 08	MOV EBX,DWORD PTR SS:[EBP+8]
7C911899	8B43 10	MOV EAX,DWORD PTR DS:[EBX+10]
7C91189C	85C0	TEST EAX,EAX
7C91189E	0F85 887E0000	JNZ ntdll.7C91972C
7C9118A4	8365 E4 00	AND DWORD PTR SS:[EBP-1C],0
7C9118A8	68 A0C0977C	PUSH ntdll.7C97C0A0
7C9118AD	E8 53F7FEFF	CALL ntdll.RtlEnterCriticalSection
7C9118B2	8365 FC 00	AND DWORD PTR SS:[EBP-4],0
7C9118B6	8B33	MOV ESI,DWORD PTR DS:[EBX]
7C9118B8	8975 E0	MOV DWORD PTR SS:[EBP-20],ESI
7C9118BB	85F6	TEST ESI,ESI
7C9118BD	74 1F	JE SHORT ntdll.7C9118DE

الشكل السابق يوضح العنوان الخاص بهذه الوظيفة ودائماً العنوان يبدأ بـ7xxxxxxx والرقم 7 ثابت أما الأرقام الأخري فمن الممكن أن تتغير من جهاز إلي أخر وإذا ذهبنا إلي العنوان الخاص بنداء هذه الوظيفة فسوف تري هذا الشكل :

Address	Hex dump	Disassembly	Comment
00401BD9	. 5A	POP EDX	
00401BDA	. 59	POP ECX	
00401BDB	. 59	POP ECX	
00401BDC	. 64:8910	MOV DWORD PTR FS:[EAX],EDX	
00401BDF	. 68 091C4000	PUSH Example.00401C09	
00401BE4	> 803D 49B0450	CMP BYTE PTR DS:[45B049],0	
00401BEB	.v 74 0A	JE SHORT Example.00401BF7	
00401BED	. 68 C4B54500	PUSH Example.0045B5C4	pCriticalSection = Ex
00401BF2	. E8 D5F7FFFF	CALL <JMP.&kernel32.LeaveCriticalSectio	LeaveCriticalSection
00401BF7	> 68 C4B54500	PUSH Example.0045B5C4	pCriticalSection = Ex
00401BFC	. E8 D3F7FFFF	CALL <JMP.&kernel32.DeleteCriticalSecti	DeleteCriticalSection
00401C01	. C3	RETN	
00401C02	.v E9 D5200000	JMP Example.00403CDC	
00401C07	.^ EB DB	JMP SHORT Example.0040	
00401C09	> 5B	POP EBX	
00401C0A	. 5D	POP EBP	

004013D4=<JMP.&kernel32.DeleteCriticalSection>

هذا العنوان هو الخاص بالقفز إلي الوظيفة المطلوبة
وهو يشير إلي جدول الـIAT

كما تري فالشكل السابق يوضح العنوان الذي يبدأ به جدول الـIAT وإذا ذهبنا إلي هذا العنوان سوف نري هذا الشكل :

```
Address   Hex dump   Disassembly                                          Comment
00401228  $- FF25 A4C14500 JMP DWORD PTR DS:[<&kernel32.CloseHandle>]    kernel32.CloseHandle
0040122E     8BC0          MOV EAX,EAX
00401230  $- FF25 A0C14500 JMP DWORD PTR DS:[<&kernel32.CreateFileA>]    kernel32.CreateFileA
00401236     8BC0          MOV EAX,EAX
00401238  $- FF25 9CC14500 JMP DWORD PTR DS:[<&kernel32.GetFileType>]    kernel32.GetFileType
0040123E     8BC0          MOV EAX,EAX
00401240  $- FF25 98C14500 JMP DWORD PTR DS:[<&kernel32.GetFileSize>]    kernel32.GetFileSize
00401246     8BC0          MOV EAX,EAX
00401248  $- FF25 94C14500 JMP DWORD PTR DS:[<&kernel32.GetStdHandle>]   kernel32.GetStdHandle
0040124E     8BC0          MOV EAX,EAX
00401250  .- FF25 90C14500 JMP DWORD PTR DS:[<&kernel32.RaiseException>] kernel32.RaiseException
00401256     8BC0          MOV EAX,EAX
00401258  $- FF25 8CC14500 JMP DWORD PTR DS:[<&kernel32.ReadFile>]       kernel32.ReadFile
0040125E     8BC0          MOV EAX,EAX
00401260  .- FF25 88C14500 JMP DWORD PTR DS:[<&kernel32.RtlUnwind>]      ntdll.RtlUnwind
00401266     8BC0          MOV EAX,EAX
00401268  $- FF25 84C14500 JMP DWORD PTR DS:[<&kernel32.SetEndOfFile>]   kernel32.SetEndOfFile
0040126E     8BC0          MOV EAX,EAX
00401270  $- FF25 80C14500 JMP DWORD PTR DS:[<&kernel32.SetFilePointer>] kernel32.SetFilePointer
00401276     8BC0          MOV EAX,EAX
00401278  $- FF25 7CC14500 JMP DWORD PTR DS:[<&kernel32.UnhandledExcep>] kernel32.UnhandledExcep
0040127E     8BC0          MOV EAX,EAX
00401280  $- FF25 78C14500 JMP DWORD PTR DS:[<&kernel32.WriteFile>]      kernel32.WriteFile
00401286     8BC0          MOV EAX,EAX
00401288  $- FF25 B8C14500 JMP DWORD PTR DS:[<&user32.CharNextA>]        user32.CharNextA
0040128E     8BC0          MOV EAX,EAX
00401290  .- FF25 74C14500 JMP DWORD PTR DS:[<&kernel32.ExitProcess>]    kernel32.ExitProcess
00401296     8BC0          MOV EAX,EAX
00401298  $- FF25 B4C14500 JMP DWORD PTR DS:[<&user32.MessageBoxA>]      user32.MessageBoxA
0040129E     8BC0          MOV EAX,EAX
004012A0  $- FF25 70C14500 JMP DWORD PTR DS:[<&kernel32.FindClose>]      kernel32.FindClose
004012A6     8BC0          MOV EAX,EAX
004012A8  $- FF25 6CC14500 JMP DWORD PTR DS:[<&kernel32.FindFirstFile>]  kernel32.FindFirstFile
004012AE     8BC0          MOV EAX,EAX
004012B0  $- FF25 68C14500 JMP DWORD PTR DS:[<&kernel32.FreeLibrary>]    kernel32.FreeLibrary
004012B6     8BC0          MOV EAX,EAX
004012B8  $- FF25 64C14500 JMP DWORD PTR DS:[<&kernel32.GetCommandLin>]  kernel32.GetCommandLin
004012BE     8BC0          MOV EAX,EAX
004012C0  $- FF25 60C14500 JMP DWORD PTR DS:[<&kernel32.GetLastErrorE>]  ntdll.RtlGetLastWin32E
004012C6     8BC0          MOV EAX,EAX
004012C8  $- FF25 5CC14500 JMP DWORD PTR DS:[<&kernel32.GetLocaleInfo>]  kernel32.GetLocaleInfo
004012CE     8BC0          MOV EAX,EAX
004012D0  $- FF25 58C14500 JMP DWORD PTR DS:[<&kernel32.GetModuleFile>]  kernel32.GetModuleFile
004012D6     8BC0          MOV EAX,EAX
004012D8  $- FF25 54C14500 JMP DWORD PTR DS:[<&kernel32.GetModuleHand>]  kernel32.GetModuleHand
004012DE     8BC0          MOV EAX,EAX
004012E0  $- FF25 50C14500 JMP DWORD PTR DS:[<&kernel32.GetProcAddres>]  kernel32.GetProcAddres
004012E6     8BC0          MOV EAX,EAX
004012E8  $- FF25 4CC14500 JMP DWORD PTR DS:[<&kernel32.GetStartupInf>]  kernel32.GetStartupInf
004012EE     8BC0          MOV EAX,EAX
004012F0  $- FF25 48C14500 JMP DWORD PTR DS:[<&kernel32.GetThreadLoca>]  kernel32.GetThreadLoca
004012F6     8BC0          MOV EAX,EAX
004012F8  $- FF25 44C14500 JMP DWORD PTR DS:[<&kernel32.LoadLibraryEx>]  kernel32.LoadLibraryEx
004012FE     8BC0          MOV EAX,EAX
00401300  $- FF25 B0C14500 JMP DWORD PTR DS:[<&user32.LoadStringA>]      user32.LoadStringA
00401306     8BC0          MOV EAX,EAX
00401308  $- FF25 40C14500 JMP DWORD PTR DS:[<&kernel32.lstrcpynA>]      kernel32.lstrcpynA
0040130E     8BC0          MOV EAX,EAX
00401310  $- FF25 3CC14500 JMP DWORD PTR DS:[<&kernel32.lstrlenA>]       kernel32.lstrlenA
00401316     8BC0          MOV EAX,EAX
00401318  $- FF25 38C14500 JMP DWORD PTR DS:[<&kernel32.MultiByteToWi>]  kernel32.MultiByteToWi
0040131E     8BC0          MOV EAX,EAX
00401320  $- FF25 C8C14500 JMP DWORD PTR DS:[<&advapi32.RegCloseKey>]    advapi32.RegCloseKey
00401326     8BC0          MOV EAX,EAX
00401328  $- FF25 C4C14500 JMP DWORD PTR DS:[<&advapi32.RegOpenKeyE>]    advapi32.RegOpenKeyExA
0040132E     8BC0          MOV EAX,EAX
00401330  $- FF25 C0C14500 JMP DWORD PTR DS:[<&advapi32.RegQueryVal>]    advapi32.RegQueryValue
00401336     8BC0          MOV EAX,EAX
00401338  $- FF25 34C14500 JMP DWORD PTR DS:[<&kernel32.WideCharToM>]    kernel32.WideCharToMul
0040133E     8BC0          MOV EAX,EAX
00401340  $- FF25 30C14500 JMP DWORD PTR DS:[<&kernel32.VirtualQuer>]    kernel32.VirtualQuery
00401346     8BC0          MOV EAX,EAX
00401348  $- FF25 D8C14500 JMP DWORD PTR DS:[<&oleaut32.SysAllocStr>]    oleaut32.SysAllocStrin
0040134E     8BC0          MOV EAX,EAX
00401350  $- FF25 D4C14500 JMP DWORD PTR DS:[<&oleaut32.SysReAllocS>]    oleaut32.SysReAllocStr
00401356     8BC0          MOV EAX,EAX
00401358  $- FF25 D0C14500 JMP DWORD PTR DS:[<&oleaut32.SysFreeStr>]     oleaut32.SysFreeString
0040135E     8BC0          MOV EAX,EAX
00401360  $- FF25 2CC14500 JMP DWORD PTR DS:[<&kernel32.Interlocke>]     kernel32.InterlockedIn
00401366     8BC0          MOV EAX,EAX
00401368  $- FF25 28C14500 JMP DWORD PTR DS:[<&kernel32.Interlocke>]     kernel32.InterlockedDe
0040136E     8BC0          MOV EAX,EAX
00401370  $- FF25 24C14500 JMP DWORD PTR DS:[<&kernel32.GetCurrent>]     kernel32.GetCurrentThr
00401376     8BC0          MOV EAX,EAX
00401378  $  53            PUSH EBX
00401379  .  83C4 BC       ADD ESP,-44
0040137C  .  BB 0A000000   MOV EBX,0A
00401381  .  54            PUSH ESP                                      pStartupinfo
00401382  .  E8 61FFFFFF   CALL <JMP.&kernel32.GetStartupInfoA>          GetStartupInfoA
00401387  .  F64424 2C 01  TEST BYTE PTR SS:[ESP+2C],1
0040138C  .v 74 05         JE SHORT Example.00401393
0040138E  .  0FB75C24 30   MOVZX EBX,WORD PTR SS:[ESP+30]
00401393  >  8BC3          MOV EAX,EBX
00401395  .  83C4 44       ADD ESP,44
00401398  .  5B            POP EBX
00401399  .  C3            RETN
0040139A     8BC0          MOV EAX,EAX
0040139C  $- FF25 20C14500 JMP DWORD PTR DS:[<&kernel32.LocalAlloc>]     kernel32.LocalAlloc
004013A2     8BC0          MOV EAX,EAX
004013A4  $- FF25 1CC14500 JMP DWORD PTR DS:[<&kernel32.LocalFree>]      kernel32.LocalFree
004013AA     8BC0          MOV EAX,EAX
004013AC  $- FF25 18C14500 JMP DWORD PTR DS:[<&kernel32.VirtualAll>]     kernel32.VirtualAlloc
004013B2     8BC0          MOV EAX,EAX
004013B4  $- FF25 14C14500 JMP DWORD PTR DS:[<&kernel32.VirtualFre>]     kernel32.VirtualFree
004013BA     8BC0          MOV EAX,EAX
004013BC  $- FF25 10C14500 JMP DWORD PTR DS:[<&kernel32.Initialize>]     kernel32.InitializeCri
004013C2     8BC0          MOV EAX,EAX
004013C4  $- FF25 0CC14500 JMP DWORD PTR DS:[<&kernel32.EnterCritic>]    ntdll.RtlEnterCritical
004013CA     8BC0          MOV EAX,EAX
004013CC  $- FF25 08C14500 JMP DWORD PTR DS:[<&kernel32.LeaveCritic>]    ntdll.RtlLeaveCritical
004013D2     8BC0          MOV EAX,EAX
004013D4  $- FF25 04C14500 JMP DWORD PTR DS:[<&kernel32.DeleteCrit>]     ntdll.RtlDeleteCritica
004013DA     8BC0          MOV EAX,EAX
```

كما تري فالشكل السابق يوضح الوظائف الخاصة بالجدول الأول الموجود في برنـامج PEBrowsePro كما يلي :

```
Table #1 (kernel32.dll):
  ImportLookupTableRVA:  0x00000000
  TimeDateStamp:         0x00000000
  ForwarderChain:        0x00000000
  NameRVA:               0x0005C740 (kernel32.dll)
  ThunkTableRVA:         0x0005C104
    Thunk01 = 0x0005C74E (0, DeleteCriticalSection)
    Thunk02 = 0x0005C766 (0, LeaveCriticalSection)
    Thunk03 = 0x0005C77E (0, EnterCriticalSection)
    Thunk04 = 0x0005C796 (0, InitializeCriticalSection)
    Thunk05 = 0x0005C7B2 (0, VirtualFree)
    Thunk06 = 0x0005C7C0 (0, VirtualAlloc)
    Thunk07 = 0x0005C7D0 (0, LocalFree)
    Thunk08 = 0x0005C7DC (0, LocalAlloc)
    Thunk09 = 0x0005C7EA (0, GetCurrentThreadId)
    Thunk10 = 0x0005C800 (0, InterlockedDecrement)
    Thunk11 = 0x0005C818 (0, InterlockedIncrement)
    Thunk12 = 0x0005C830 (0, VirtualQuery)
    Thunk13 = 0x0005C840 (0, WideCharToMultiByte)
    Thunk14 = 0x0005C856 (0, MultiByteToWideChar)
    Thunk15 = 0x0005C86C (0, lstrlenA)
    Thunk16 = 0x0005C878 (0, lstrcpynA)
    Thunk17 = 0x0005C884 (0, LoadLibraryExA)
    Thunk18 = 0x0005C896 (0, GetThreadLocale)
    Thunk19 = 0x0005C8A8 (0, GetStartupInfoA)
    Thunk20 = 0x0005C8BA (0, GetProcAddress)
    Thunk21 = 0x0005C8CC (0, GetModuleHandleA)
    Thunk22 = 0x0005C8E0 (0, GetModuleFileNameA)
    Thunk23 = 0x0005C8F6 (0, GetLocaleInfoA)
    Thunk24 = 0x0005C908 (0, GetLastError)
    Thunk25 = 0x0005C918 (0, GetCommandLineA)
    Thunk26 = 0x0005C92A (0, FreeLibrary)
    Thunk27 = 0x0005C938 (0, FindFirstFileA)
    Thunk28 = 0x0005C94A (0, FindClose)
    Thunk29 = 0x0005C956 (0, ExitProcess)
    Thunk30 = 0x0005C964 (0, WriteFile)
    Thunk31 = 0x0005C970 (0, UnhandledExceptionFilter)
    Thunk32 = 0x0005C98C (0, SetFilePointer)
    Thunk33 = 0x0005C99E (0, SetEndOfFile)
    Thunk34 = 0x0005C9AE (0, RtlUnwind)
    Thunk35 = 0x0005C9BA (0, ReadFile)
    Thunk36 = 0x0005C9C6 (0, RaiseException)
    Thunk37 = 0x0005C9D8 (0, GetStdHandle)
    Thunk38 = 0x0005C9E8 (0, GetFileSize)
    Thunk39 = 0x0005C9F6 (0, GetFileType)
    Thunk40 = 0x0005CA04 (0, CreateFileA)
    Thunk41 = 0x0005CA12 (0, CloseHandle)
```

فإذا قمت بتطابق الجدول السابق مع الشكل النـاتج من برنـامج Ollydbg سوف تـري أن الوظائف متطابقة وبذلك نكون قد إنتهينا من شرح الـImport Table.