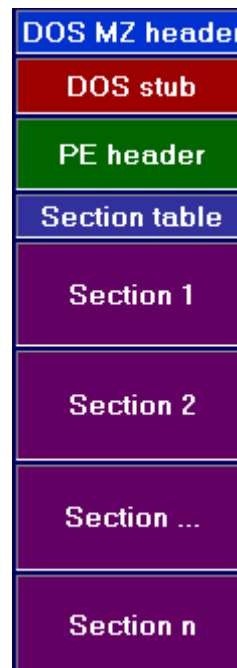


Introduction :

في هذا الدرس سنتجد شرحا عن ملفات PE لما لها من أهمية كبيرة في عمليات ال reverse engineering .
للأمانة العلمية ، فهذا الدرس أو المقال هو ترجمة وتلخيص لمقال للكاتب Iczelion .

Okay , let's start:

إن PE ترمز إلى Portable Executable ...وهي صيغة "طبيعية" (native) لملفات windows32 . إن معنى التنقل (portable) هو إن أي PE loader على أي جهاز win32 سوف يقوم بتشغيل هذا الملف بغض النظر عن نوع المعالج المستخدم. إن هذا النوع من الملفات شائع جدا على كثير من الأنظمة. يتكون أي ملف PE من الأقسام العامة التالية كما يتضح بالصورة:



DOS MZ header & DOS stub

إن أول قسمين غير مهمين لنا لكن يمكنني القول أنهما مسئولان عن جملة this program cannot run in dos mod ولعلك لاحظت هذه الجملة كثيرا عند محاولتك لتشغيل ملف باستخدام الدوس. سنتحدث قليلا عن القسم الأول... يسمى هذا القسم بـ IMAGE_DOS_HEADER . هذا القسم يحتوي قيمتين فقط مهمتين لنا هما e_lfanew و e_magic . إن e_magic تحتوي على ال "MZ" string وعند قراءة ملف ال PE من قبل ال loader فانه يبحث عن هذه القيمة ويقارنها مع IMAGE_DOS_SIGNATURE فإذا تطابقا فال DOS HEADER صالح (valid). أما e_lfanew . فهذه ال structure تحتوي على ال offset ل PE Header

PE Header

القسم الثالث هو ال PE Header . عندما يتم تحميل الملف بواسطة PE loader من على نظام يتعرف على PE files كـ win32 فإن هذا ال loader يقرأ ال offset ل PE Header من القسم الأول ثم "يقفز" إن صح التعبير عن القسم الثاني ويتوجه إلى الثالث مباشرة. مما يجب عليك معرفته هو إن تنظيم ال data إلى sections يتم على أساس ال attribute ، فبيانات ال read-only لوحدها وبيانات ال system لوحدها وهكذا مع hidden, archive الخ. إياك إن تظن إن التقسيم يتم على أساس منطقي أي ال data و ال code وغيرها كل لوحده. جدير بالذكر إن عملية التقسيم يقوم بها ال loader وتسمى هذه العملية عادة بـ mapping the sections into memory (لعلك لاحظت ال memory map في olly) إن هذا القسم يكون شكله كالتالي:

IMAGE_NT_HEADERS STRUCT

Signature dd ?

FileHeader IMAGE_FILE_HEADER <>

OptionalHeader IMAGE_OPTIONAL_HEADER32 <>

IMAGE_NT_HEADERS ENDS

لاحظ إن ال PE header في الواقع عبارة عن structure .
والآن سأفسر ما يظهر بالأعلى:

IMAGE_NT_HEADERS هو اسم ال PE header structure
Signature هي dword تحوي القيم التالية 00h, 45h, 00h, h أي "PE" متبوعة بصفرين.
FileHeader هو structure تحتوي على معلومات كعدد الأقسام ، ال machine التي يعمل عليها الملف وهكذا.
OptionalHeader لا تتدع بالاسم فهذا الجزء ليس "اختياريا" بل هو أساسي وبه عدة معلومات ستذكر لاحقاً.

عند قراءة الملف ، إذا كانت قيمة *Signature* تساوي "PE" متبوعة بصفرين فيتم تصنيف الملف على أنه ملف PE صالح (valid)
: FileHeader

والآن دعنا نتحدث عن ثاني جزء في ال PE header وهو *FileHeader* الذي يظهر بالشكل التالي:

```
IMAGE_FILE_HEADER STRUCT
Machine WORD ?
NumberOfSections WORD ?
TimeDateStamp dd ?
PointerToSymbolTable dd ?
NumberOfSymbols dd ?
SizeOfOptionalHeader WORD ?
Characteristics WORD ?
IMAGE_FILE_HEADER ENDS
```

Machine : نوع ال platform أي ال CPU التي يعمل عليها الملف...للمعالجات انتل تكون كالتالي
IMAGE_FILE_MACHINE_I386
NumberOfSections : كما هو واضح من الاسم يحتوي على عدد الأقسام .
TimeDateStamp : غير مهم
PointerToSymbolTable : غير مهم
NumberOfSymbols : غير مهم
SizeOfOptionalHeader : غير مهم
Characteristics : بعض المعلومات ، كنوع الملف (exe أم dll)

الجزء الأخير هو *OptionalHeader* ... قبل الحديث عن أجزائه دعنا نشرح ال RVA
هذه الحروف الثلاثة ترمز لـ relative virtual address ، ببساطة ال RVA هو المسافة من نقطة مرجعية.
على سبيل المثال إذا تم تحميل ملف ال PE عن العنوان h400000 في الفراغ الوهمي virtual address (حاول إن تتخيله!!)
وبدأ البرنامج بالتنفيذ (execution) عند العنوان الوهمي h401000 ، عندها يمكننا القول إن البرنامج قد بدأ عند RVA=1000h
وفي هذا المثال فإن RVA قيمته 1000 نسبة للنقطة المرجعية وهي بداية ال virtual address (الفراغ الوهمي)
لكن لماذا يستخدم ملف ال PE RVAs ؟؟ إن هذا يؤدي إلى تخفيف الحمل load عن ال loader. (إذا أردت إن تعرف كيف يتم ذلك
فراجع المقال الأصلي لـ iczelion)

: OptionalHeader

والآن نأتي لمكونات *OptionalHeader* : (لم أجد الشكل العام له...لكنه مشابه لشكل ال *FileHeader* مع اختلاف العناصر)
AddressOfEntryPoint : هذا هو قيمة RVA التي تشير إلى عنوان أول instruction سوف يتم تنفيذها.
ImageBase : هذا هو القيمة "المفضلة" التي يستخدمها ال loader . مثلاً لو كانت هذه القيمة h400000 فإن ال loader سوف يحاول تحميل (loading) الملف عند هذا العنوان...لكن لو كانت هذه المنطقة محجوزة من قبل تطبيق آخر فسيضطر ال loader للبحث عن منطقة خالية أخرى.
SectionAlignment : لو كانت قيمته h1000 فسيبدأ كل section من عنوان عند مضاعفات h1000 فمثلاً لو كان أول section يبدأ عند h4001000 فالثاني سيبدأ عند h4002000 وهكذا.
FileAlignment : نفس فكرة السابق لكن يختلف عنه.
MajorSubsystemVersion/MinorSubsystemVersion : هذا هو win32 subsystem version وعادة يحمل القيمة 4.0
SizeOfImage : الحجم الكلي لـ PE image في الذاكرة.
SizeOfHeaders : حجم ال headers+section table أي أنه يساوي حجم الملف ناقص حجم جميع ال sections
Subsystem : دعك منه
DataDirectory : هذه عبارة عن مصفوفة لـ *IMAGE_DATA_DIRECTORY* .

: section table

إن section table هو array of structures كل structure تحوي معلومات عن كل قسم في PE file كـ attribute, the file offset, virtual offset. لو كان هناك 3 أقسام فإن ال section table يحوي 3 قواعد (structures) ، وهو يسمى بـ IMAGE_SECTION_HEADER ويكون كالتالي :

IMAGE_SIZEOF_SHORT_NAME equ 8

IMAGE_SECTION_HEADER STRUCT

Name1 db IMAGE_SIZEOF_SHORT_NAME dup (?)

union Misc

PhysicalAddress dd ?

VirtualSize dd ?

ends

VirtualAddress dd ?

SizeOfRawData dd ?

PointerToRawData dd ?

PointerToRelocations dd ?

PointerToLinenumbers dd ?

NumberOfRelocations dw ?

NumberOfLinenumbers dw ?

Characteristics dd ?

IMAGE_SECTION_HEADER ENDS

VirtualAddress : قيم RVA لهذا القسم...فإن كانت قيمته h3000 وكان ال loader قد بدا عند h400000 فإن هذا القسم (section table) سيتم تحميله عند h4003000 .

SizeOfRawData : حجم ال section data أي قسم البيانات. (rounded up to the next multiple of file alignment)

PointerToRawData : يحتوي على offset لبداية هذا القسم (section table)

Characteristics : بعض الرايات (flags) ...مثلا هل يحتوي القسم على executable code, initialized data, uninitialized data هل يمكن القراءة منه أو الكتابة عليه؟

: IMPORT TABLE

دعنا بداية نتعرف على import function ..أنها دالة غير موجودة في البرنامج...كل ما هو موجود هو معلومات عنها أما الدالة نفسها فموجودة في احد ملفات ال DLL. (بالنسبة لدوال API فملفات ال DLL موجودة في مجلد system32 في مجلد وندوز الرئيسي) هل تذكر آخر عنصر في ال optional header ؟ نعم انه DataDirectory ...كما قلنا فهي عبارة عن مصفوفة تسمى IMAGE_DATA_DIRECTORY بها 16 عنصر إن DataDirectory يحتوي على ال locations and sizes للبيانات الهامة في الملف. كما هو واضح في الصورة ..(بظهر عنصر 14 فقط)

Member	Info inside
0	Export symbols
1	Import symbols
2	Resources
3	Exception
4	Security
5	Base relocation
6	Debug
7	Copyright string
8	Unknown
9	Thread local storage (TLS)
10	Load configuration
11	Bound Import
12	Import Address Table
13	Delay Import
14	COM descriptor

العناصر الملونة بالأصفر هي فقط المهمة... كل عنصر من العناصر الـ 16 له الاسم IMAGE_DATA_DIRECTORY (متبوعاً باسم العنصر) وله الشكل العام التالي

IMAGE_DATA_DIRECTORY STRUCT

VirtualAddress dd ?

Size dd ?

IMAGE_DATA_DIRECTORY ENDS

VirtualAddress : هو الـ RVA للـ DataDirectory

Size : يحوي الحجم بالبايت.

إذا كنت تريد أن تعرف أي يقع Base relocation (مثلاً) فعليك ضرب قيمة حجم IMAGE_DATA_DIRECTORY أي 8 بايت بـ 5 ثم تضيف الناتج إلى عنوان IMAGE_DATA_DIRECTORY. مثال : عنوان IMAGE_DATA_DIRECTORY هو 4001000 وحجمه 8 بايت. إذن Base relocation يقع عند عنوان 4001000+40 أي 4001040.

والآن دعنا نتحدث عن import table. عنوان هذا القسم موجود في خانة VirtualAddress في العنصر الثاني من الـ DataDirectory (أي في import symbols). إن الـ import table هو مصفوفة IMAGE_IMPORT_DESCRIPTOR. لو كان الـ PE File يستورد دوال من 10 ملفات DLL مختلفة فإن هذه المصفوفة تحوي 10 IMAGE_IMPORT_DESCRIPTOR. هذه هي الشكل العام لها :

IMAGE_IMPORT_DESCRIPTOR STRUCT

union

Characteristics dd ?

OriginalFirstThunk dd ?

ends

TimeDateStamp dd ?

ForwarderChain dd ?

Name1 dd ?

FirstThunk dd ?

IMAGE_IMPORT_DESCRIPTOR ENDS

أول عنصر هو الـ union وهو يقدم اسماً مستعاراً (alias) للـ OriginalFirstThunk. لذا يمكن اعتبار الـ Union بمثابة Characteristics. بعبارة أخرى، هذا العنصر يحوي عنوان RVA للـ IMAGE_THUNK_DATA والتي بدورها تشير لأحد الـ IMAGE_IMPORT_BY_NAME. دعنا نفسر ما سبق ذكره.

هناك عدة دوال مستوردة import function

كل import function لها IMAGE_IMPORT_BY_NAME

كل IMAGE_IMPORT_BY_NAME لها RVA (فلنسمه 1 RVA)

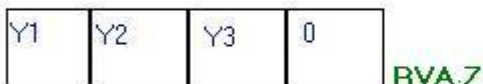
كل 1 RVA يتم تخزينه في IMAGE_THUNK_DATA

كل IMAGE_THUNK_DATA بدورها لها RVA (فلنسمه 2 RVA)

يتم ترتيب جميع الـ RVAs في مصفوفة تنتهي بالقيمة 0... الـ RVA للمصفوفة الناتجة (فلنسمه 3 RVA) يتم وضعه في OriginalFirstThunk. إن لم تستوعب ذلك فانظر إلى الصورة التالية.



allko



OriginalFirstThunk=Z

FirstThunk=Z

إن كل IMAGE_IMPORT_BY_NAME تبدو بالشكل التالي:

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

Name1 : غير مهم
Hint : غير مهم

نعود إلى IMAGE_IMPORT_DESCRIPTOR STRUCT لشرح العناصر:

TimeStamp : في المرحلة الحالية ، غير مهم

ForwarderChain : في المرحلة الحالية ، غير مهم

Name1 : يحوي على RVA لاسم ال DLL

FirstThunk : مشابه لـ *OriginalFirstThunk* ... (هل ترى الشرح في الصورة؟ انه يطبق أيضا على ال *FirstThunk* وبالتالي في النهاية سنحصل على مصفوفتين بدل من واحدة...الأولى *OriginalFirstThunk* والثانية *FirstThunk* .

على سبيل المثال لو كان PE file يستورد 10 دوال من kernel32.dll فإن *Name1* في IMAGE_IMPORT_DESCRIPTOR سيحوي على RVA لل string المسماة "kernel32.dll". وسوف يكون هنالك 10 IMAGE_THUNK_DATAs في ال

FirstThunk و *OriginalFirstThunk* ومثلهم في

لكن السؤال هو...لم نحتاج إلى مصفوفتين متماثلتين (*FirstThunk* و *OriginalFirstThunk*)؟؟

عند تحميل الملف فإن ال loader ينظر إلى IMAGE_IMPORT_BY_NAMES and IMAGE_THUNK_DATAs ليحدد عناوين الدوال المستوردة. ومن ثم يقوم باستبدال IMAGE_THUNK_DATAs (الخاصة بـ *FirstThunk*) ، يستبدلها بالعناوين الحقيقية لتلك الدوال المستوردة. أي إن الدالة المنتهية بـ 0 والتي يشير إليها *OriginalFirstThunk* لا تتغير بينما مثلتها المنتهية بـ 0 والتي يشير إليها *FirstThunk* تتغير محتوياتها (قيم Y بالصورة) إلى العناوين الحقيقية للدوال المستوردة. ممم...وما الهدف من ذلك؟

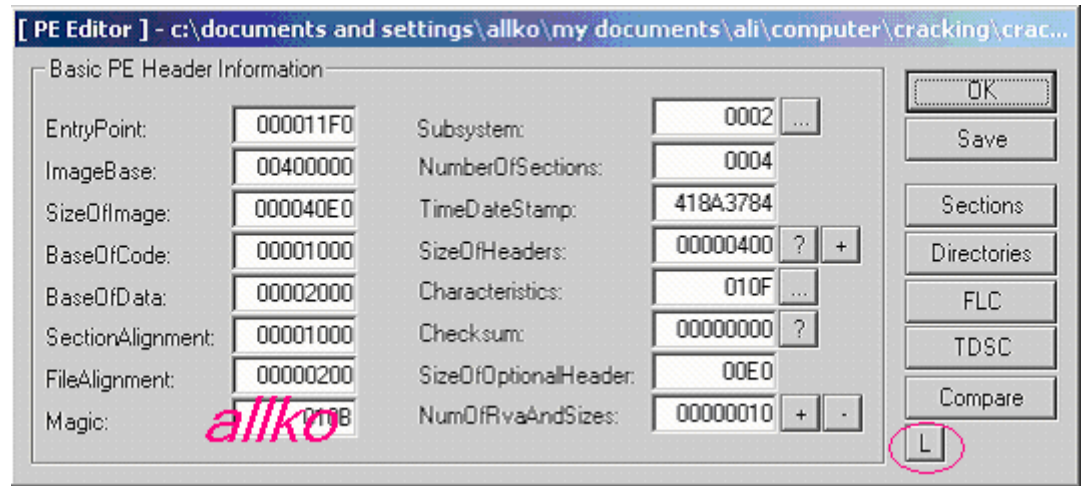
قلنا بان الدوال المستوردة يتم استدعائها باسمها...لكن هناك بعض الدوال لا تستدعى بالاسم بل بالترتيب (ordinal) ...في هذه الحالة لن يكون هناك IMAGE_IMPORT_BY_NAME لأنه ببساطة لا يوجد اسم للدالة..عوضا عن ذلك فإن

IMAGE_THUNK_DATA لتلك الدالة المستوردة بالموضع سيحتوي على ترتيب (موضع) هذه الدالة في الجزء الأصغر منه ، و على 1 في موضع ال MSB. مثلا إذا كانت الدالة تستورد بترتيبها ، وكان ترتيبها h123 فإن IMAGE_THUNK_DATA لهذه الدالة سيكون 80001234h (عند التحويل إلى بالنظام الثنائي ستصبح رقم طويل وفي أقصى اليسار (MSB) الرقم 1)...

والآن لنفترض أننا نريد إن نحصل على قائمة بأسماء جميع الدوال المستوردة في ملف PE ما ...الخطوات هي:

- 1- تأكد إن الملف هو ملف PE صالح valid PE
- 2- من ال DOS header اذهب إلى PE header
- 3- احصل على عنوان data directory من OptionalHeader
- 4- اذهب إلى العنصر الثاني في data directory (أي : Import symbols) واستخرج VirtualAddress
- 5- استخدم تلك القيمة للذهاب إلى IMAGE_IMPORT_DESCRIPTOR
- 6- تحقق من قيمة *OriginalFirstThunk*. إذا كانت لم تكن صفرا ، فاتبع عنوان RVA للذهاب إلى تلك المصفوفة(المنتهية بـ 0) أما إذا كانت صفرا فاستخدم القيمة في *FirstThunk* بدلا من *OriginalFirstThunk* (السبب في ذلك هو إن بعض ال linkers تضع 0 دائما في *OriginalFirstThunk* رغم إن هذا يعتبر bug)
- 7- لكل عنصر في المصفوفة(المنتهية بـ 0) تحقق من قيمة MSB : إذا كانت 1 إذن هذه الدالة مستوردة بالترتيب. ونحصل على الترتيب من الجزء الأصغر (1234h80001234 => h)
- 8- إذا كان ال MSB يساوي 0 فاذهب إلى تلك المصفوفة(المنتهية بـ 0) والقيمة التي ستجدها هي عبارة عن RVA لـ IMAGE_IMPORT_BY_NAME اذهب إليها لتحصل على اسم الدالة.
- 9- الآن اذهب إلى العنصر التالي في المصفوفة (المنتهية بـ 0) وكرر الخطوات لتحصل على اسم للدالة...تابع إلى إن تجد الصفرة(مللت وأنا أقول : المنتهية بـ 0 ، أخيرا وصلنا إلى هذا ال 0): ووصلنا إلى الصفرة يعني انتهاء المصفوفة أي أننا حصلنا على أسماء جميع الدوال المستوردة من ذلك ال DLL.
- 10- اذهب إلى IMAGE_IMPORT_DESCRIPTOR التالي وكرر نفس الخطوات لاستخراج الأسماء لملف DLL التالي .

والآن كنطبق على الدرس شغل LordPE ثم اختر PE Editor واختر أي ملف exe لتظهر شاشة مشابهة للتالي:



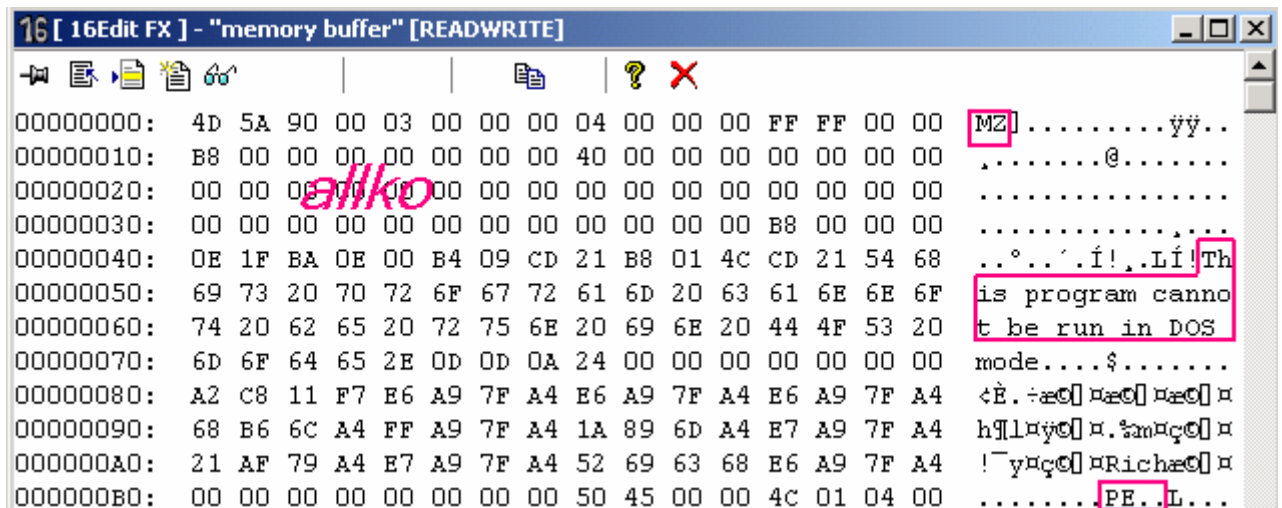
لاحظ أن معظم هذه الخصائص قمنا بشرحها والباقي ليس ذو أهمية كبيرة...
اضغط على الزر L والآن ستظهر لك list بها مكونات هذا ال PE بدئا ب DOS Header وانتهائا ب DataDirectory..
تمعنه جيدا فهو يقوم بربط أفكارك معا.
حسننا لقد قلنا أن كل ملف PE مكون من عدة sections... sections التي تلي ال 4 الأولى (أي بعد DOS MZ header و DOS stub و PE Header و section table) نستطيع أن نجدهم بالضغط على زر sections لتظهر نافذة مشابهة للتالي :

[Section Table]					
Name	VOffset	VSize	ROffset	RSize	Flags
.text	00001000	000006B0	00000400	00000800	60000020
.data	00002000	00000028	00000C00	00000200	C0000040
.idata	00003000	00000384	00000E00	00000400	C0000040
.rsrc	00004000	000000E0	00001200	00000200	50000040

كما ترى هناك 4 أقسام هي text. و data. و idata. و الأخير.rsrc. لل resources.
بقي ال Import Table... اختر Directories ليظهر ال IT بعناصره ال 16. اضغط على حرف L بجانب أي عنصر لتلقي عليه نظرة عن قرب... حسننا اضغط على H بجانب أي منها كما بالصورة (زر H الأول لن يظهر لك شيئا... حسننا لا بأس جرب غيره... ما نريده هو عرض الملف ولا يهم أي زر تضغط..)

	RVA	Size			
ExportTable:	00000000	00000000	...	L	H
ImportTable:	000020A0	00000050	...	L	H
Resource:	00004000	00002B30	...	L	H
Exception:	00000000	00000000	...	L	H

سيظهر لك الملف بهيئة hex... اصعد للأعلى إلى بداية الملف لتر التالي :



هل تلاحظ حرفي MZ ؟ هذا هو قسم DOS MZ Header ...ثم هل تلاحظ تلك الجملة التي تحدثنا عنها؟ هذا هو قسم DOS stub ...ثم هلا تلاحظ حرفي PE المتبوعين بصفين؟ هذا هو قسم PE Header. وهكذا مع البقية.

إلى هنا نكون قد انتهينا من هذا المقال...أمل إن أكون قد وفقت في عمليتي التلخيص والترجمة..إن كانت هناك أخطاء علمية في الترجمة فالرجاء إبلاغي بذلك.

Do you know:

ما هي البرامج الحرة ؟
هي طريقة متبعة لحماية الملكية الفكرية لا تقوم على احتكار المعلومة بل على نشرها خرج بالفكرة Richard M. Stallman مؤسس مؤسسة البرامج الحرة Free Software Foundation (www.fsf.org) يمكن لأي كان أن يحصل على الملف المصدري للينكس أو أن يعمل نسخ من البرنامج ويوزعها مجاناً أو يبيعها.

Outroduction :

Finally I finished this article. IT takes me about 8 hours to translate and summarize the original one by Icelion.

I advice you to read this material thoroughly and if you didn't understand it from the first time , don get anxious coz this is normal. For me it required me to reread it two times to understand. As usual , your criticisms are welcomed.

Thx:

DiDi, Icelion , Xacker and Dahaka.

23/2/2006 6:35 pm

There are many crackers, but only one allko
allko34@hotmail.com