	Saggio sul PEditor 1.6 Spieghiamo TUTTO il formato PE	
02/11/2000	by "AndreaGeddon"	
	UIC's Home Page	Published by Quequero
Qualcuno dovrebbe fare un tute...	*Certo che dovevamo fare a botte ma quando? E non mi dovevi portare la bottiglia di grappa? E la cioccolata? E poi non dire che sono io a non mantenere le promesse :))) Torniamo a noi: codesto brutto animal (tale andrewgheddone o simile) cerca in un modo piuttosto prolisso di farci capire cosa sia il PE dandoci istruzioni su come usare un programma che a LUI sembra buono....Ma come si sa, le sue idee sono opinabili. In pratica questa persona che io NON conosco vuole insegnarci a coltivare le patate facendoci andare in giro col trattore... Vabbuò, se lo dice lui io poco ci posso fare, e neanche me la sento di obiettare visto che codesta persona potrebbe risentirsi, ad ogni modo, se volete confondervi ancora di più le idee potete leggere questo pessimo (IMHO) tutorial, formattato anche male.....:)))))) Bravo Andreucolo :))))*	..su come usare questo maledetto form
	Home page se presente: www.andreageddon.8m.com / www.andreageddon.com E-mail: andreageddon@hotmail.com IRC: irc.azzurra.it / irc.tin.it #crack-it	
Difficoltà	( )**NewBies** ( )**Intermedio** (x)**Avanzato** (x)**Master**	

In questo testo verrà spiegato tutto il formato PE. Ogni spiegazione sarà corredata da esempi pratici (per esempio sulla Import Table) per capire a fondo come funziona e come è fatto questo benedetto PE.

Saggio sul PEditor 1.6

Spieghiamo TUTTO il formato PE

Written by **AndreaGeddon**

Introduzione

Questa non è una guida all'unpacking, ma al PEditing. Tuttavia conoscendo a fondo il PE vi risulterà più facile comprendere e smontare i packers.

Tools usati

Un editor esadecimale (per andare ad esaminare i bytes nell'eseguibile)
Il PEditor (dato che questo testo è ispirato a tale programma)
Notepad (io ho quello del win98 prima edizione) Winzip 7 - Usati come cavie

URL o FTP del programma

http://playtools.cjb.net

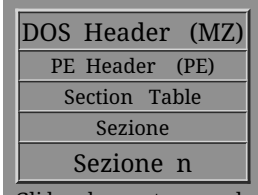
Notizie sul programma

Dopo la chiusura ufficiale del progetto ProcDump ecco che arriva un degno sostituto. Ho notato alcune diversità nei dump effettuati dal ProcDump e dal PEditor, sono arrivato alla conclusione che il ProcDump è superiore in certe cose, il PEditor in altre. Non credo che si possa in definitiva dire quindi quale è il migliore, di sicuro averli entrambi vi garantirà un ottimo supporto nelle vostre unpacking wars.

Essay

Vi riporto il testo che ho scritto origiariamente per OndaQuadra, la e-zine degli hackmaniaci (le tabelle le ho rifatte visto che da ascii ad
html non vengono bene):
-

---+
-o DISCLAIMER
---+
Le informazioni presenti in questo testo sono a scopo puramente didattico. L'autore e l'E-Zine non si ritengono responsabili dell'uso che farete di queste informazioni (e francamente se ne fregano).
---+
-o OVERVIEW
---+
Ho trovato questo bellissimo programma, che è una sorta di ProcDump ma più evoluto e con molte più funzioni. Ho pensato quindi di descriverne tuuutte le funzioni e di spiegare tutti i campi che ci permette di modificare in un eseguibile. Ovviamente verranno spiegate anche la teoria del Manual Unpacking e il formato PE, senza i quali la spiegazione del prog non starebbe in piedi. Man mano che tratterò vari punti di questo programma, ne spiegherò anche la relativa teoria. Il seguente documento inoltre può essere usato come riferimento per tutti i programmi di PE-Editing che volete.
+-- INTRO -- Informazioni sull'uso del programma e sua reperibilità, accenni al formato PE.
|
+-- CENNI DI TEORIA SUL MEMORY MAPPING E SUL FORMATO PE
|
+-- LA SCHERMATA INIZIALE -- Commento degli header Optional & File e funzioni principali
|
+-- LA SEZIONE "SECTIONS" -- Vediamo le sezioni da vicino
| |
| +-- Relativo Menù del tasto destrto
|
+-- LA BARRA DEI COMANDI SULLA DESTRA
|
+-- LA SEZIONE "DIRECTORY" -- Accurata descrizione dell'import/export table etc...
|
+-- NOTE
|
+-- LIBRARY, THANKS, LINKS AND E-MAIL
---+
-o INTRO
---+
A cosa serve il PEditor? Serve per manipolare file eseguibili del formato standard PE (Portable Executable) della piattaforma Win32, ma è anche un potente Memory Tool che permette di dumpare i processi che volete, ottenerne informazioni, killarli, etc. In pratica, molti dei maggiori sistemi di protezione utilizzano la tecnica del Crypting (o Packing), che consiste nel criptare il file eseguibile in modo da impedire agli smanettoni di andare a ficcare il naso nel programma per modificarlo a proprio piacimento. Questi programmi si autodecriptano all'esecuzione, quindi si troveranno in memoria già belli che decriptati, ed è la che arriviamo noi col PEditor: non dobbiamo fare altro che prendere il file dalla memoria e salvarlo su disco, per ottenere una copia del file eseguibile già decriptata e funzionante e pronta per essere manipolata da noi. Peccato che in realtà non è mai così semplice, visto che spesso c'è bisogno di rimettere a posto alcune cosette per far partire il programma. Il PEditor se ben usato ci permette di fare tutto questo.
Il programma lo trovate su Player's Tools:
http://playtools.cjb.net
gli autori sono M.O.D. e Yoda.
---+
-o CENNI DI TEORIA SUL MEMORY MAPPING E SUL FORMATO PE
---+
Ogni file eseguibile ha una sua struttura ben precisa. Tale struttra varia a seconda della piattaforma per la quale è stato scritto il software. Quella di cui ci interessemo qui è la struttra del PE, il COFF (Common Object File Format) degli eseguibili standard della piattaforma win32. Tale struttura ci servirà anche per capire come il programma viene caricato in memoria. Vediamola in breve:



Gli header contengono le informazioni sul file eseguibile stesso, la section table contiene le informazioni delle varie sezioni, e le sezioni sono i veri e propri blocchi di dati e codice accomunati dalle stesse proprietà. Di solito ci sono delle sezioni standard, tipo:
.text o .CODE - Sezione di codice eseguibile
.data - Sezione di data
.rsrc - Sezione delle risorse, tipo bitmap, dialogs etc.
.reloc - Sezione della relocation table
e altre un pò meno comuni. Dare uno sguardo alle sezioni è sempre utile, non solo per farci un'idea del programma, ma anche per ottenere preziose informazioni: insomma, se trovate la sezione .Shrink non avete difficoltà a capire che avete a che fare con uno Shrinker :-).
Ora abbiamo una idea di come è fatto un file eseguibile PE, vediamo brevemente come viene caricato, eseguito e mappato in memoria. Quando eseguiamo un file, il PE loader legge gli header (MZ e PE), da lì legge le varie informazioni sul file che ci si sta apprestando a caricare (informazioni che ritroveremo

e spiegheremo in seguito), quindi le varie sezioni vengono mappate in memoria. In genere l'indirizzo di inizio per la mappatura è di 00401000 per un eseguibile (cioè la sua image-base + CodeBase, vedi dopo). Gli eseguibili protetti da packers o crypters sono leggermente diversi: il file fisico contiene le sezioni criptate, quindi non possiamo andarle a disassemblare o modificare, a meno che non conosciamo a priori l'algoritmo di crypting. Durante l'esecuzione il file si auto decripta e si mappa in memoria già decriptato e funzionante. L'idea è quindi quella di copiare il contenuto della memoria (relativo al file eseguibile decriptato) e salvarlo su disco. Otterremmo così senza sforzo il nostro eseguibile decriptato. Come immaginate la cosa non è così semplice! Ci sono vari problemi che si incontrano nel fare questa operazione (che viene chiamata "Dumping"): alcuni dovuti al SO, altri dovuti alla bastardaggine dei programmatori che l'hanno implementato. Per esempio, il Windox usa il meccanismo della paginazione per gestire la memoria, cioè le sezioni del file eseguibile vengono mappate a pagine di 4kb. Il problema è che per risparmiare memoria non sempre sono presenti TUTTE le pagine del programma che vogliamo dumpare (ad esempio alcune pagine sono state swappate sul disco), quindi usando il comando PAGE del softice dovremmo andare a vedere la lista delle pagine con i relativi attributi Present/Not Present, poi con il comando PAGEIN dovremmo caricare fisicamente in memoria le pagine non presenti. A quel punto abbiamo in memoria il file completo e pronto per essere dumpato. Una volta dumpato poi il file ancora non funzia, quindi dobbiamo andare a vedere cosa c'è che non va, e qui succedono i casini, perchè bisogna vedere come hanno implementato i programmatori il sistema di Crypting: ad esempio una cosa molto comune è la manomissione della Import Table. Addirittura si potrebbero anche incontrare problemi quando si va a fare il PAGEIN. Ecco perchè andremo a studiarci il PEditor: conoscendo a fondo tutte ciò che ci permette di fare il programma possiamo riuscire ad attaccare e risolvere i vari crypter.

---+

-o LA SCHERMATA INIZIALE

---+

Quando avviate il PEditor vi appare la schermata iniziale con molti campi che corrispondono ognuno ad un valore specifico. Caricate un programma eseguibile (io per esempio carico il WinZip32 v7.00) e vedrete gli edit box riempirsi di numerelli :-). Commentiamoli tutti!
Piccola precisazione:
RVA = Relative Virtual Address, cioè indirizzo RELATIVO all'image base (del tipo 00001234)
VA = Virtual Address, cioè indirizzo virtuale che troviamo in memoria (del tipo 00401234)
Tenete Presente che in ogni campo del PEditor troveremo quasi sempre RVA, e questi valori non indicano direttamente la posizione dell'oggetto nel file fisico. Per ottenere il vero offset bisogna calcolarselo attraverso il FLC. In questo testo per comodità ho scelto programmi in cui l'rva coincide sempre con l'offset fisico (fenomeno spiegato in seguito), però non dimenticate mai di considerare l'rva come un valore che deve sempre essere convertito in file offset.

-- Entry Point --

Questo valore ci dà l'indirizzo fisico dell'Entry Point, cioè dell'indirizzo da cui il programma inizia l'esecuzione. Nel mio caso è 00055C20. Ho detto indirizzo FISICO in quanto questo valore corrisponde alla locazione nel file eseguibile, cioè al 55C20 = 351264-esimo byte ci sarà il punto dove il programma deve iniziare l'esecuzione. L'indirizzo di memoria corrispondente sarà ottenuto sommando 55C20 al valore datoci dall'Image Base (però non è sempre così), di cui parleremo fra poco.

-- Image Base --

Nel mio caso vale 00400000 ma praticamente quasi tutti gli eseguibili hanno questo valore. L'image base non è altro che il valore da cui viene iniziata la mappatura del file eseguibile, cioè è l'indirizzo a partire dal quale viene messo il file in memoria. Se debuggate spesso non avrete potuto fare a meno di notare che nei file eseguibili le istruzioni che seguite si trovano sempre ad indirizzi tipo 004xxxxx. Appunto questo valore ve lo dà l'image base. Tale valore cambia a seconda del formato dell'eseguibile, cioè se EXE o DLL... etc. L'image base per le DLL per esempio è più vario, ma generalmente l'indirizzo è del tipo X0000000, per i tipi SYS in genere è 00010000 etc.

-- Base of Code --

Qui trovo il valore 00001000. Anche questo è quasi sempre uguale per tutti gli eseguibili. La sezione di codice di un eseguibile inizia all'indirizzo 00401000, che appunto è dato da Image Base + Base of Code.

-- Base of Data --

68000. Questo valore invece cambia di volta in volta. Infatti è dato dalla grandezza delle sezioni di codice precedenti: nel caso del winzip abbiamo le sezioni .text e INIT_TEX che finiscono appunto all'indirizzo 68000. Da qui in poi ci sono sezioni di dati.

-- Size of Image --

La grandezza dell'immagine, cioè la grandezza che verrà mappata in memoria. Nel mio caso è FA000, che corrisponde a 1'024'000. Il size fisico dell'eseguibile è invece di 983'040 bytes. Questa differenza è data dal fatto che gli eseguibili devono rispettare dei valori di allineamento (che vederemo dopo), sia per gli indirizzi VA (in memoria), sia per l'allineamento FISICO nel file (vedi Section Alignment e File Alignment). Tali valori di allineamento significano che una sezione può iniziare e finire solo ad un indirizzo che sia un multiplo del valore di allineamento.

-- Size of Headers --

Come intuibile, ci indica la grandezza degli headers (MZ+PE) e della Object Table (la table che descrive le sezioni). Se questo valore è = a 1000 avremo un allineamento perfetto tra Virtual Offset e Raw Offset, cioè VA - Image Base = Offset fisico (perchè la mappatura delle sezioni in memoria inizia da 00401000!). Se invece il size è diverso da 1000, allora per calcolarci l'offset fisico dobbiamo usare la formula: VA - Image Base - Base of Code + Size of Headers = Offset Fisico.

-- Section Alignment --

Nel mio caso (ma molto spesso) 1000. Le sezioni vengono mappate in memoria secondo multipli di questo valore. Per esempio, se una sezione è grande 1234, in memoria non verrà mappata uno spazio di 1234 ma uno spazio di 2000 (primo multiplo di 1000 dopo 1234). Quando arrivo alla sezione sulle sezioni (sorry) farò alcuni esempi e vedrete che capirete al volo.

-- File Alignment --

Come prima, solo che questo allineamento è relativo al file fisico. Anche nel file fisico le sezioni vengono scritte a multipli di questo valore (che di solito è 200, ma varia parecchio). Ovviamente se mettiamo un valore di allineamento troppo grande, sprecheremo dello spazio nel file fisico.
-- SubSystem --
Il sistema richiesto dal programma. Nel mio caso 0002. Questo campo può avere i seguenti valori:

0000h __Unknown
0001h __Native
0002h __Windows GUI
0003h __Windows Character
0005h __OS/2 Character
0007h __Posix Character

-- Machine Type --
Il tipo di macchina per cui è stato scritto il sw. Nel mio caso 14C. I valori di questo campo sono:

0000h __sconosciuto
014Ch __80386
014Dh __80486
014Eh __80586
0162h __MIPS Mark I (R2000, R3000)
0163h __MIPS Mark II (R6000)
0166h __MIPS Mark III (R4000)

quindi il programma in esame girerà su tutte le macchine 386+.
-- Number of Sections --
Indovinate ???????
-- Time Date Stamp --
Data e ora dalla creazione del file (creazione fatta dal linker).
-- Pointer To Symbol Table --
Usato per il debugging.
-- Number of symbols --
Usato per il debugging.
-- Size of Optional Header --
L'optional header è quello contiene le informazioni di Base of code, Size of Image etc. Questo campo vi dice il suo size (nota che l'optional header però non contiene solo i campi che vediamo dal PEditor)
-- Characteristcs --
Contiene vari flag, ad esempio per specificare se il PE considerato è una DLL o un EXE.
----------+
-o LA SEZIONE DELLE SEZIONI
----------+
Nella schermata iniziale c'è un pulsante etichettato "Sections". Premetelo e vi aprirà la finestra con la lista e descrizione delle sezioni. Io ad esempio carico il 5° corso newbies di Quequero, e mi si presenta la seguente tabella:

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	0000322B	00001000	00003400	00000400	60000020
.rdata	000007F6	000007F6	00000800	00003800	40000040
.data	000008E4	00006000	00000800	00004000	C0000040
.idata	000009BA	00007000	00000A00	00004800	C0000040
.rsrc	00000854	00008000	00000A00	00005200	40000040
.reloc	00000430	00009000	00000600	00005C00	42000040

Bella vero? Qui troviamo tutte le informazioni sulle sezioni.
-- Section --
IL primo campo (Section) ci dice il nome
delle varie sezioni. Di solito iniziano con un ".", ma non è sempre così. Il nome delle sezioni spesso è anche relativo al contenuto: "text" indica una sezione di codice (anche .CODE), .data o simili indicano sezioni di dati, .rsrc indica la sezione delle risorse (bitmap, icone, dialogs etc...), .reloc indica la sezione delle Rilocazioni (è questa la traduzione di Relocations?). Qui è il caso di spendere due parole sulle rilocazioni. Premetto che in un file eseguibile questa sezione è completamente inutili, visto che nel 99% dei casi un eseguibile non subisce MAI una rilocazione. Ma cos'è una rilocazione? E' un fenomeno che troviamo spesso nelle DLL. Quando un processo carica una dll, il codice della dll viene mappato su un certo indirizzo xxx. Se però il programma carica diverse dll, si troverà a mappare di nuovo al tale indirizzo xxx, dove è già presente una dll. Quindi il sistema operativo deve caricare questa dll in un altro indirizzo yyy. A causa di questa rilocazione anche tutti i puntatori della dll vengono spostati!
Per esempio se normalmente avete una definizione di una stringa, in asm vedrete che il puntatore a questa stringa sarà 00401234. Quando la dll verrà mappata, all'indirizzo 00401234 troverete la stringa. Se però la dll è stata rilocata, anche questo indirizzo verrà cambiato per puntare alla stringa mappata nel nuovo spazio yyy. Bene, la sezione .reloc contiene tutti questi indirizzi fissi, così quando la dll viene caricata, il sistema operativo controlla questi indirizzi; se sono già occupati in memoria (da una dll precedente) allora li alloca da un'altra parte. Detto così può sembrare un discorso campato in aria, ma

quando vi trovate a dover patchare delle dll le rilocazioni vi provocano problemini non indifferenti!
-- Virtual Size --
Questi valori indicano la grandezza reale della sezione in memoria. Ad esempio la sezione .text risulta grande 322B bytes. Nota che questi valori sono REALI, e non soddisfano nessun allineamento.
-- Virtual Offset --
Indica l'RVA a partire dal quale sarà mappata in memoria la sezione. Per la sezione .text il valore è di 1000 e non zero. Perchè? Perchè questo 1000 è il famoso "Base of code". Quindi la prima sezione (.text) partirà dal VA 00401000. Adesso ritorniamo al discorso degli allineamenti. Per la sezione .text io parto da 00401000 e mappo 322B bytes. Quindi essendo le sezioni contigue, dovrei aspettarmi la prossima sezione mappata da 00401000 + 322B = 0040422B. Il valore di allineamento delle sezioni, però, qui è di 1000, quindi la prossima sezione comincerà dal primo multiplo di 1000 disponibile dopo 0040422B, cioè al VA 00405000 (RVA = 5000). E così via tutte le sezioni. L'ultima sezione comincia all'RVA 9000 ed è lunga 430 bytes. Quindi in tutto l'ultimo byte sarà il 9430°. Sempre per l'allineamento, la fine di questa sezione va a finire a A000, che è il valore del Size Of Image.
-- Raw Size --
La grandezza che le sezioni occupano nel file fisico. Anche questo valore è correlato ad un allineamento, quello dato da File Alignment, che di solito è pari a 200. In effetti tutti i size sono multipli di 200.
La prima sezione ad esempio abbiamo visto che VIRTUALMENTE è grande 322B. Il primo multiplo di 200 disponibile dopo tale valore è 3400, quello del raw size. Lo spazio che va da 322B a 3400 viene riempito con taanti zeri (469 decimale per l'esattezza). Bene, immaginate che questo si ripete per tutte le sezioni di tutti gli eseguibili (e tutti i file con formato PE) che avete sull'HD! Già! Avete un hard disk pieno di inutili zeri! Che ci volete fare...
-- Raw Offset --
L'indirizzo fisico dal quale è mappata la sezione nel file. Nel nostro caso per la sezione .text abbiamo un offset di 400 e non di zero. 400 infatti è il valore che ritroviamo in "Size of Headers". Quindi gli header MZ+PE sono grandi 400 bytes, e da qui inizieranno le sezioni. La raw size di .text è 3400, quindi la prossima sezione sarà a 3400+400 = 3800. Come vedete qui tutto combacia con l'allineamento a 200.
A questo punto prendiamo il raw offset dell'ultima sezione (5C00), aggiungiamogli il raw size dell'ultima sezione (600). Risultato: 6200. Aggiungiamo ancora 200 (del file alignment) e otteniamo 6400, che in decimale vale 25600, esattamente il valore della grandezza del file fisico.
-- Characteristcs --
Questo campo descrive il tipo di sezione (dati, eseguibile, etc.). I valori possono essere i seguenti

000000020h Sezione di codice
000000040h Sezione di dati inizializzati
000000080h Sezione di dati non inizializzati
040000000h La sezione non deve essere messa in cache
080000000h La sezione non è paginabile
100000000h La sezione è condivisa
200000000h Sezione eseguibile
400000000h Sezione in lettura
800000000h Sezione in scrittura

e possono essere combinati. Ad esempio la sezione .text ha il valore 60000020, che è una combinazione di 000000020 + 40000000 + 20000000, cioè Sezione di codice, eseguibile e leggibile. La sezione .data, invece, è una combinazione di 80000000 + 40000000 + 00000040, cioè dati inizializzati, lettura e scrittura. Questi attributi sono molto importanti perchè ci danno un idea precisa delle sezioni che abbiamo davanti.
--== IL MENU' DEL TASTO DESTRO ==--
Se cliccate col destro su una sezione avrete il segente menù:
>>Edit Section
Vi aprirà un nuovo DialogBox con tutti i campi della sezione, e ve li potrete modificare a vostro piacimento. Notate il tasto Char Wizard: questo apre una apposia finestra che vi permette di modificare automaticamente le flag di "Characteristcs".
>>Add a section in the PE Header
Per aggiungere una sezione.
>>Delete the section in the PE Header
Per cancellare la sezione selezionata.
>>Copy the section to HD
Salva la sezione in un file su HD.
>>Move the section to HD
Salva la sezione su HD e poi la cancella dall'eseguibile.
>>Copy a section from HD to EOF
Prende una sezione salvata su disco e la inserisce alla fine del file (dopo le altre sezioni).
>>DumpFixer (RS=VS & RO=VO)
Dopo aver unpackato un file eseguibile molto probabilmente dovrete rimettere a posto i valori Size e Offset sia Virtuali che Raw. Prima ce lo facevamo a parte col procdump, ora con il PEditor è automatico! Questa operazione si chiama "Riallineamento del PE", e a questo punto dovrebbe esservi chiaro perchè dobbiamo farla: dopo aver dumpato un programma è probabile che le sezioni contengano dei dati non giusti sulla Raw Size; infatti dopo che avete unpackato il file, le sezioni risultano più grandi di quanto lo erano prima nel file fisico compresso, quindi dobbiamo cambiare i relativi descrittori.
>>Set the characteristics to E0000020
Setta automaticamente le flag della sezione a CODICE, ESEGUIBILE, LETTURA, SCRITTURA.
>>Truncate at the start of this section
Cancella tutti i byte dalla sezione selezionata in poi.

>>Custom truncate
Cancella tutti i byte da un offset specificato in poi.
----------+
-o LA BARRA DEI COMANDI SULLA DESTRA
----------+
Descriviamo ora i comandi (quelli in colonna a destra).
-- Browse --
Se questo non l'avete capito da soli spegnete il pc e dategli fuoco.
-- Tasks --
L'elenco dei task in esecuzione. Per ogni task avete i dati, cioè Process ID, Image Base, Size of Image
e Owner. Premendo su un task con il tasto destro avrete il seguente menù:

dump (full) dump (partial) dump (Sections)
view process infos load into PEditor
terminate process
refresh

>> dump full: vi permette di dumpare l'intero processo in automatico
>> dump partial: vi permette di dumpare una area di memoria a piacimento, scegliendo gli offset iniziale
e finale del dump.
>> dump sections: apre un dialogbox in cui avete una lista delle sezioni che potrete dumpare
separatamente, compreso l'header PE.
>> view process infos: apre un dialog che contiene i seguenti campi:
Usage
Th32ProcessId
Th32DefaultHeapId
Th32ModuleId
CntThreads
Th32ParenProcessId
PcPriClassBase
>> load into PEditor: attenzione per gli assuefatti al ProcDump! Questa opzione carica solo il file exe
relativo al processo, e non i dati relativi al processo stesso! Cioè, in procdump voi con process
infos caricavate le info sulle sezioni a runtime, con PEditor invece non potete.
>> terminate process: killa un processo
>> refresh: refresha tutti i valori
-- ! --
Refresha tutti i valori di tutti i campi.
-- Section Split --
Dumpa in automatico tutte le sezioni separate, ognuna in un file diverso che verrà messo nella dir
corrente. Attenzione perchè non vi viene chiesta nessuna conferma!
-- Break'n Enter --
Questa funzione serve per breakare su un indirizzo qualsiasi di un programma. Come funziona? Per esempio
prima andavamo a scriverci a mano un CCh all'indirizzo per trapparlo con softice tramite BPINT 3, ora con
break'n enter settiamo un BPINT 3 da softice, premiamo il tasto RUN e popperemo direttamente sullo
indirizzo. Non solo! In sice avremo nella prompt window un riquadro che ci dice che byte dobbiamo
ripristinare all'indirizzo al posto del CC! Non è stramitico ??!!
-- FLC --
Sta per File Location Calculator. Praticamente vi richiede un valore che può essere uno dei seguenti
campi: VA, RVA, Offset esadecimale, Offset decimale; tale valore verrà convertito negli altri campi
e vi vengono pure indicati i primi bytes che si trovano all'offset in quel file.
-- Checksum --
Calcola il checksum corretto del file e vi dice quale è invece il checksum attualmente memorizzato nel
file (di solito 0). C'è anche la possibilità di correggereil checksum in automatico tramite l'apposito
tasto "correct it".
-- Rebuilder --
Il rebuilder serve per ricostruire (o meglio, provare a ricostruire) la import table. Spesso dopo il
dump un programma può avere la Import Table (IT) compromessa, quindi per far funzionare l'eseguibile
dumpato avremo bisogno di ricreargli la IT. Questa cosa si può fare in diversi modi, e il Rebuilder è
uno di questi! La comodità del rebuilder è che lo fa in automatico, però non sempre riesce (anzi, io
finora non sono mai riuscito a ricostruire una IT semplicemente col rebuilder). Cmq vediamo come funzia:
O-O Wipe Relocation Section
Questa opzione permette di eliminare la Relocation Section, permettendo così direcuperare un pò di
spazio. La sezione Relocation negli eseguibili, come abbiamo visto, è praticamente inutile, visto che
gli eseguibili non subiscono mai le rilocazioni.
O-O Realign File (Only Win 9x)
Normal: Riduce il valore di allineamento del file, in modo da guadagnare spazio.
HardCore: Elimina anche il Dos Stub (una parte dell'header MZ, in genere quella che contiene il
messaggio "Questo prog ha bisogno di winzozz...) per guadagnare ancora altro spazio.
Notate che la riduzione di spazio ottenuta con il riallineamento non è sempre significativa.
O-O Rebuild Import Table
Fast: Cerca nell'import table le stringhe delle funzioni importate.
Carefully: Cerca nell'intero file le stringhe delle funzioni importate.

Force functionsearching: Forza la ricerca nell'intero file delle stringhe delle funzioni importate
anche se c'è già l'OriginalFirstThunk.
Nota-- Il discorso del table rebuilding è abbastanza complesso, qui ho dato solo la semplice
descrizione delle opzioni, i dettagli della Import Table verranno approfonditi più avanti nella
Sezione Directory/Imports.
Make PE Header Win NT/2k compatible: cerca di rendere il file compatibile con il Win NT controllando
il Size of Image, il File Size, Null Sections e fa il Section Overlap (boh!).
ecco fatto. Ora scegliete le opzioni più opportune e provate a ricostruire tutte le IT che volete. Vi
ricordo che se possibile le IT è meglio ricavarle in fase di dumping, o con il ProcDump (Rebuild New IT)
o ve la dumpate a mano. E' possibile che vi ritroverete ad aver aggiustato una IT, e scoprirete che il
file dumpato e riparato funzionerà solo sul vostro PC. Anche questo spiacevole inconveniente verrà
approfondito più avanti quando parleremo della IT.
-- Apply changes --
Se avete modificato un valore, con questo pulsante potete fare l'update.
-- About --
La finestra di about :-) Notate tra i ringraziamenti il nome del nostro caro Kill3xx!
-- Exit --
Indovinate ?

---+

-o LA SEZIONE DIRECTORY

---+

Nella schermata principale avete il pulsante "Directory". Se ci cliccate verrà aperto un nuovo dialog
pieno di nuove funzioni e valori da commentare (gasp! ma quando finisce 'sto testo??). Questa sarà la
parte più difficile e importante, visto che parleremo della Import/Export Table e dei problemi connessi.
Cominciamo dal trafiletto di dwords: per ogni campo abbiamo sia RVA che SIZE. RVA ci dà l'indirizzo di
mappatura dell'oggetto in considerazione, SIZE ci dà la grandezza dell'oggetto stesso. In questa ultima
versione del PEditor hanno anche aggiunto i tips, tenete il mouse su un campo e vi dirà la posizione
fisica all'interno del file. Per esempio nel WinZip 8, vado sul campo Import Table e mi appare il tip:
"position of this entry: 00000170h (e_lfanew + 80h) related to: .rdata"
dove e_lfanew è un membro della struttura IMAGE_DOS_HEADER, membro che contiene il puntatore all'header
PE. 80h è la posizione del puntatore all'IT all'interno della struttura Directory, e related to .rdata
vuol dire che il VA si trova all'interno della sezione rdata (ehm.. questi tips stanno anche nella
schermata principale, avevo dimenticato di menzionarli :-)).
-- Export Table --
Nel programma in esame (WinZip 8) questo campo ha valori nulli. Di solito un exeguibile non ha delle
funzioni esportate. Per avere un esempio sottomano prendiamo il file Wz32.dll dalla dir del winzip.
Ora per la Export Table avremo l'RVA 0002FC60 e il SIZE 0000000149. Siccome siamo curiosi prendiamo un
hex-editor e andiamo a vedere con i nostri occhi questa esprt table. Carichiamo il wz32.dll, quindi
prendiamo l'rva 0002FC60 e col FLC lo convertiamo in raw offset. In questo caso il raw offset coincide
con 2FC60, a causa del perfetto allineamento. Ecco cosa troviamo:

```
02FC60 00 00 00 00 - E4 EE FD 38 - 00 00 00 00 - 0A FD 02 00 ................
02FC70 01 00 00 00 - 0D 00 00 00 - 0D 00 00 00 - 88 FC 02 00 ................
02FC80 BC FC 02 00 - F0 FC 02 00 - C0 83 01 00 - 60 83 01 00 ................
02FC90 50 83 01 00 - 46 E9 00 00 - E0 E8 00 00 - 10 72 01 00 ................
02FCA0 70 71 01 00 - A0 9C 01 00 - 80 97 01 00 - 60 C9 01 00 ................
02FCB0 00 C9 01 00 - 10 BF 01 00 - B0 C9 01 00 - 13 FD 02 00 ................
02FCC0 22 FD 02 00 - 32 FD 02 00 - 37 FD 02 00 - 42 FD 02 00 ................
02FCD0 52 FD 02 00 - 58 FD 02 00 - 63 FD 02 00 - 67 FD 02 00 ................
02FCE0 70 FD 02 00 - 81 FD 02 00 - 90 FD 02 00 - 99 FD 02 00 ................
02FCF0 00 00 01 00 - 02 00 03 00 - 04 00 05 00 - 06 00 07 00 ................
02FD00 08 00 09 00 - 0A 00 0B 00 - 0C 00 77 7A - 33 32 2E 64 ..........wz32.d
02FD10 6C 6C 00 44 - 72 61 67 41 - 70 70 65 6E - 64 46 69 6C ll.DragAppendFil
02FD20 65 00 44 72 - 61 67 43 72 - 65 61 74 65 - 46 69 6C 65 e.DragCreateFile
02FD30 73 00 57 5A - 35 36 00 75 - 6E 63 6F 6D - 70 72 65 73 s.WZ56.uncompres
02FD40 73 00 75 6E - 63 6F 6D 70 - 72 65 73 73 - 5F 69 6E 69 s.uncompress_ini
02FD50 74 00 75 6E - 7A 69 70 00 - 75 6E 7A 69 - 70 5F 69 6E t.unzip.unzip_in
02FD60 69 74 00 7A - 69 70 00 7A - 69 70 5F 69 - 6E 69 74 00 it.zip.zip_init.
02FD70 7A 69 70 6C - 61 62 65 6C - 44 69 73 6B - 65 74 74 65 ziplabelDiskette
02FD80 00 7A 69 70 - 6D 65 6D 63 - 6F 6D 70 72 - 65 73 73 00 .zipmemcompress.
02FD90 7A 69 70 73 - 70 6C 69 74 - 00 7A 69 70 - 77 69 70 65 zipsplit.zipwipe
02FDA0 44 69 73 6B - 65 74 74 65 - 00                       Diskette.
```

(nota che sono riportati nella notazione intel, quindi ad esempio la seconda dword non è E4EEFD38 ma
il suo contrario 38FDEEE4)
questi sono i 329 Bytes della Export Table (149h, il Size). Come vedete la seconda metà è piena di
quelli che sembrano i nomi delle funzioni. Okei, diamo un significato a questa manciata di bytes.
La Export Table è strutturata nel modo seguente:
1. Directory Table (informazioni sulla ET stessa) (prime 10 dwords)
2. Address Table (puntatori alle funzioni nella dll)
3. Name Pointer Table (puntatori ai nomi delle funzioni)
4. Ordinal Table (struttura ordinale delle funzioni)
5. Name Strings (nomi delle funzioni)
bene, abbiamo tutti i dati per interpretare la nostra ET. Partiamo dalla Directory Table, che è formata
dalle prime 10 dwords, che identificano la struttra seguente:

VALORE	DWORD NELLA ET DI ESEMPIO
EXPORT FLAGS	00 00 00 00 1°
TIME/DATE STAMP	E4 EE FD 38 2°
MAJOR VERSION | MINOR VERSION	00 00 | 00 00 3°
NAME RVA	0A FD 02 00 4°
ORDINAL BASE	01 00 00 00 5°
# EAT ENTRIES	0D 00 00 00 6°
# NAME PTRS	0D 00 00 00 7°
ADDRESS TABLE RVA	88 FC 02 00 8°
NAME PTR TABLE RVA	BC FC 02 00 9°
ORDINAL TABLE RVA	F0 FC 02 00 10°

()- Export Flags: attualmente questo valore non è mai usato, sta sempre settato a zero.
()- Time/Date Stamp: data e ora di creazione dell'export.
()- Major/Minor version: due words che raramente sono usate.
()- Name RVA: punta alla stringa che rappresenta il nome della DLL
Nel nostro caso punta a 0002FD0A, che è l'indirizzo della stringa wz32.dll (contare per credere).
()- Ordinal Base: indica il numero di partenza della EAT (export address table), normalmente è a 1.
()- # EAT Entries: il numero delle funzioni esportate (nel nostro caso 0D = 13dec).
()- # Name Ptrs: indica il numero di funzioni nella Name Ptr Table, cioè abbiamo 13 nomi di funzioni.
Non sempre questo valore coincide con il numero dell funzioni effettivamente presenti nella ET.
Indica solo il numero dei NOMI delle funzioni esportate.
()- Address Table RVA: indica la struttra che contiene i puntatori al codice delle funzioni.
Nel nostro caso punta alla seguente struttura:

02FC88 C0 83 01 00 - 60 83 01 00 .....
02FC90 50 83 01 00 - 46 E9 00 00 - E0 E8 00 00 - 10 72 01 00 .....
02FCA0 70 71 01 00 - A0 9C 01 00 - 80 97 01 00 - 60 C9 01 00 .....
02FCB0 00 C9 01 00 - 10 BF 01 00 - B0 C9 01 00 .....

Come vedete sono tutti puntatori (RVA) a zone di codice precedenti a quella della ET. Ad esempio avremo che la prima dword 000183C0 identifica l'RVA del codice che effettivamente contiene la prima funzione esportata, la seconda dword 00018360 identifica il puntatore alla seconda funzione effettiva di codice e così via.
()- Name ptr table RVA: indica la struttura che contiene i nomi delle funzioni importate.
Nel nostro caso punta a 0002FCBC, che è la seguente struttura:

02FCBC 13 FD 02 00 .....
02FCC0 22 FD 02 00 - 32 FD 02 00 - 37 FD 02 00 - 42 FD 02 00 .....
02FCD0 52 FD 02 00 - 58 FD 02 00 - 63 FD 02 00 - 67 FD 02 00 .....
02FCE0 70 FD 02 00 - 81 FD 02 00 - 90 FD 02 00 - 99 FD 02 00 .....

Anche questi sono puntatori (sempre RVA) ai nomi delle funzioni esportate, prendiamo la prima dword: 0002FD13 punta alla stringa "DragAppendFile" che è il nome della prima funzione esportata, la seconda dword punta a 0002FD22, cioè la stringa "DragCreateFiles", e così via per tutte e 13 le funzioni esportate.
()- Ordinal Table RVA: indica la struttura degli Ordinali (li vedremo fra poco).
Di nuovo vediamo che punta a 02FCF0, ecco cosa ci trovate:

02FCF0 00 00 01 00 - 02 00 03 00 - 04 00 05 00 - 06 00 07 00 .....
02FD00 08 00 09 00 - 0A 00 0B 00 - 0C 00 .....

i numeri da 0000 a 000C (gli ordinali delle 13 funzioni). Beh, prima vi ho tenuto in sospeso sul fatto di questi ordinali, ora è il momento di fare luce! Le funzioni possono essere esportate tramite i nomi, come in questo caso, o anche soltanto tramite i loro numeri ordinali, senza dover usare i nomi. Sicuramente vi sarà capitato di debuggare qualche programma che nella sua Import Table non ha nomi di funzioni carini tipo "CheckTheSerial" o "CheckIfCdIsCopied", ma avrete solo quei noiosissimi "Ordinal 0160h" e così via. Questo è dovuto proprio al fatto che il programma fa riferimento a una libreria che ha le export in Ordinal Only. Il programma identifica le funzioni da richiamare semplicemente dal numerello a 16 bit (assegnato univocamente a ogni funzione), numero che viene appunto usato da GetProcAddress per ricavarsi l'indirizzo della funzione voluta.

-- Import Table --

Se fin qui gli argomeni trattati vi sono sembrati difficili, spegnete il computer e andate a giocare con le macchinine (o con le bambole), perchè stiamo per affrontare l'argomento più ostico di tutto questo testo, e francamente anche il più importante, quello che vi ritroverete spesso a dover fronteggiare quando ingaggerete battaglie disperate con crypters assurdi. Cmq fin qui è stato tutto facile, giusto? Vedrete che alla fine se spiegato bene anche questo argomento diventerà facile come gli altri.
Tanto per cominciare mi serve una cavia adatta: qual'è un file con una IT semplice che abbiamo tutti? il Notepad.exe! Andiamo nella Directory Table e per la IT troviamo:
Import table: RVA: 00006000 SIZE: 8C
quindi abbiamo indirizzo e grandezza. Ora 6000 è un RVA, quindi col FLC andiamocelo a convertire in raw offset, ed otteniamo... 6000 (allineamento perfetto).

B0 61 00 00 - ED 6C 45 37 - FF FF FF FF - 8A 65 00 00
F0 63 00 00 - 18 61 00 00 - B3 C2 1F 37 - FF FF FF FF
9C 67 00 00 - 58 63 00 00 - CC 61 00 00 - EB 6C 45 37
FF FF FF FF - 92 6B 00 00 - 0C 64 00 00 - B8 60 00 00
72 C2 3E 35 - FF FF FF FF - F6 6C 00 00 - F8 62 00 00
C0 62 00 00 - ED 6C 45 37 - FF FF FF FF - 7A 6D 00 00
00 65 00 00 - A0 60 00 00 - EB 6C 45 37 - FF FF FF FF
DA 6D 00 00 - E0 62 00 00 - 00 00 00 00 - 00 00 00 00
00 00 00 00 - 00 00 00 00 - 00 00 00 00

questi sono i 140 (8Ch) bytes. Per ora non compaiono nomi di funzioni o dll.
Partiamo dalla Directory Table, che come al solito contiene le informazioni sulla IT stessa. Dobbiamo prendere in considerazione le prime 5 dwords, che ci identificano i seguenti valori:

VALORE:	DWORD NELLA IT DI ESEMPIO
Union: IMPORT FLAGS or OriginalFirstThunk	B0 61 00 00
TIME/DATE STAMP	ED 6C 45 37
FORWARDER CHAIN	FF FF FF FF
NAME RVA	8A 65 00 00
IMPORT ADDRESS TABLE RVA (FirstThunk)	F0 63 00 00

anche le dword successive vanno considerate a gruppi di 5, compreso l'ultimo gruppo di 5 dwords che serve per chiudere la Directory dell'IT.
()- Import Flags
Questo dovrebbe rientrare nel campo "characteristics", ma in realtà la dword è una union tra il valore Characteristics e OriginalFirstThunk. Essendo il campo characteristics sempre a zero, la dword contiene il valore dell'OriginalFirstThunk, 000061B0, che punta alla struttura degli indirizzi ai nomi delle funzioni importate:

7A 65 00 00 - 68 65 00 00 - 20 65 00 00 - 4E 65 00 00
3C 65 00 00 - 2E 65 00 00 - 00 00 00 00

questi valori dunque puntano ai nomi delle funzioni (la NULL indica il termine della struttura), e questi indirizzi di nomi faranno parte degli OriginalFirstThunk (ne parleremo dopo).
E le Characteristics? La union è come una struttura, solo che i membri di tale struttura possono essere utilizzati solo uno alla volta. Quindi il la union in questione può contenere O le characteristics O OriginalFirstThunk. Notate che sono 6 funzioni importate, e che l'ordine dei loro nomi non coincide con l'ordine dei loro puntatori (cosa che noterete se andate a vedere l'IT dal PEditor).
In realtà questi indirizzi sono dei puntatori ad altre strutture, cioè:
Struttura IMAGE_IMPORT_BY_NAME:
word HINT
byte NAME1
infatti prendiamo ad esempio il primo puntatore, 0000657A, punta alla stringa:
0000657A 6C 00 53 68 65 6C 6C 45 78 65 63 75 74 65 41 00 l.ShellExecuteA
in cui i primi due byte sono la word HINT (006C) che rappresenta l'indice della funzione nell'ET della dll da cui stiamo importando la funzione, e poi il nome vero e proprio, ShellExecuteA in forma di array di byte terminati dallo 0.
()- Time Date Stamp
La solita dword con la data e ora di creazione della IT.
()- Forwarder Chain
Nel nostro caso è settato a -1, e quindi non ci sono forwarders. Cos'è un forwarder? Una dll può esportare un simbolo che non è definito nella dll stessa, ma è importato da un'altra dll. In tal caso il simbolo si dice Forwarded. Ora sorge il problema che nel FirstThunk di un Forwarded noi abbiamo l'entry point della funzione, però il loader non ha modo di verificare se questo entry point è corretto dato che non si trova realmente nella dll da cui stiamo importando. Quindi l'entry point dell forwarded deve essere sempre fixato, e tale fix può avvenire tramite il ForwarderChain. Lo troviamo come indice nella lista dei thunks, in cui la funzione alla posizione indicizzata è una funzione esportata in forward, e il FirstThunk a tale posizione è l'indice della prossima funzione importata. La catena continua fino a trovare il valore -1, che indica che non ci sono più forwarders.
()- Name RVA
Questo punta al nome della dll importata. Abbiamo un bel 0000658A. Andiamo a cercare anche questo rva nel file (non serve la conversione in raw offset perchè come abbiamo visto l'allineamento è perfetto) e ci troviamo la stringa:
0000658A 53 48 45 4C 4C 33 32 2E 64 6C 6C 00 SHELL32.dll
()- Import Address Table RVA
Abbiamo il valore 000063F0. Questo valore punta ad un array di dwords, l'ultima delle quali è nulla ed indica la fine della tabella.
Nel nostro caso a 000063F0 troviamo:

FF 6A D1 7F - 0F 28 CD 7F - 66 16 CE 7F - 7C 84 CB 7F
EB C6 CC 7F - 7B 42 D1 7F - 00 00 00 00

6 dowrd più l'ultima NULL. Queste dwords ci danno gli indirizzi delle funzioni importate per ogni dll, però non le import all'interno del file (OriginalFirstThunk), ma delle import functions vere e proprie (FirstThunk).

Ho come l'impressione di non essere stato molto chiaro. Allora continuaimo l'esempio pratico, tanto per eliminare tutti i dubbi. Sempre nella Import Table che abbiamo appena esaminato, torniamo all'indirizzo della IT e troviamo:

6000 B0 61 00 00 - ED 6C 45 37 - FF FF FF FF - 8A 65 00 00 Primo Modulo
F0 63 00 00

18 61 00 00 - B3 C2 1F 37 - FF FF FF FF Secondo Modulo
9C 67 00 00 - 58 63 00 00

- CC 61 00 00 - EB 6C 45 37 Terzo Modulo
FF FF FF FF - 92 6B 00 00 - 0C 64 00 00

- B8 60 00 00 Quarto Modulo
72 C2 3E 35 - FF FF FF FF - F6 6C 00 00 - F8 62 00 00

C0 62 00 00 - ED 6C 45 37 - FF FF FF FF - 7A 6D 00 00 Quinto Modulo
00 65 00 00

- A0 60 00 00 - EB 6C 45 37 - FF FF FF FF Sesto Modulo
DA 6D 00 00 - E0 62 00 00

- 00 00 00 00 - 00 00 00 00 NULL
00 00 00 00 - 00 00 00 00 - 00 00 00 00

bene, così divisi sono più leggibili. Ogni 5 dword abbiamo la struttura IMAGE_IMPORT_DESCRIPTOR, questa struttura contiene le informazioni sulla dll importata e le sue relative funzioni. Il primo modulo lo abbiamo già visto, era SHELL32.DLL, ora passiamo al secondo. Andiamo subito alla terza dword, otteniamo l'indirizzo 0000679C. Andiamo a cercare nel file l'offset e troviamo la stringa KERNEL32.dll, quindi abbiamo ricavato il nome della dll. Prendiamo la prima dword del secondo modulo e abbiamo il puntatore 00006118, troviamo il seguente array:

. EE 66 00 00 - FC 66 00 00
06 67 00 00 - E4 66 00 00 - CC 66 00 00 - 1A 67 00 00
24 67 00 00 - BE 66 00 00 - DA 66 00 00 - 4E 67 00 00
5C 67 00 00 - 6C 67 00 00 - 7E 67 00 00 - 90 67 00 00
B8 65 00 00 - A4 65 00 00 - 96 65 00 00 - 96 66 00 00
B2 66 00 00 - A2 66 00 00 - 66 66 00 00 - 86 66 00 00
7A 66 00 00 - 3C 66 00 00 - 5A 66 00 00 - 48 66 00 00
12 66 00 00 - 30 66 00 00 - 20 66 00 00 - EA 65 00 00
08 66 00 00 - FC 65 00 00 - 30 67 00 00 - DC 65 00 00
CA 65 00 00 - 10 67 00 00 - 40 67 00 00 - 00 00 00 00

questi sono tutti i puntatori a strutture HINT, cioè strutture formate come già visto da una dword che indica il numero ordinale della funzione e da una stringa. Quindi i puntatori avranno il significato:

000066EE 5F 00 44 65 6C 65 74 65 46 69 6C 65 41 00 _.DeleteFileA
indice ordinale: 5F nome: DeleteFileA
000066FC D1 02 5F 6C 63 72 65 61 74 00 .._lcreat
indice ordinale: 02D1 nome: _lcreat
etc.....

abbiamo i nomi, quindi torniamo nel secondo modulo e andiamo a vedere la quinta dword, 00006358, che ci dà l'array (IAT):
F5 19 FA BF - AD 07 FA BF
BA 74 F7 BF - D7 07 FA BF - FE 09 FA BF - CC B5 F9 BF
D0 49 F7 BF - B4 48 F7 BF - BB B5 F9 BF - B4 48 F7 BF
D4 71 F7 BF - 07 7D F7 BF - 6F 7F F7 BF - BE 72 F7 BF
AD 77 F7 BF - 16 77 F7 BF - F8 D4 F8 BF - 32 73 F7 BF
2B 08 FA BF - 5F 6E F7 BF - D0 77 F7 BF - 40 7E F7 BF
A9 73 F7 BF - C0 64 F7 BF - 84 72 F7 BF - 57 7B F7 BF
DB 7A F7 BF - 6F 73 F7 BF - 6B 51 F8 BF - C6 20 F8 BF
5F 33 F9 BF - F8 72 F7 BF - 1B 6E F7 BF - 3D 6E F7 BF
DA C5 F8 BF - E4 74 F7 BF - D7 6D F7 BF - 00 00 00 00

e questi sono gli indirizzi veri e propri delle funzioni importate. Non sempre il numero dei nomi di funzioni è effettivamente uguale al numero degli indirizzi della IAT.
Possiamo quindi riassumere la seguente corrispondenza:

OriginalFirstThunk	Nome Funzione	FirstThunk
000066EE	DeleteFileA	BFFA19F5
000066FC	_lcreat	BFFA07AD

etc...
ecco quindi spiegato tutto il meccanismo di import. Quando eseguiamo un file, il PE viene caricato in

memoria, poi il PE loader trova nella IT il nome della dll importata, quindi va a vedere gli Original-FirstThunk, da lì ricava il nome della funzione, e poi va a trovare nella IAT l'indirizzo della funzione in esame. Prendiamo il caso di DeleteFileA. Disassemblo il notepad.exe e vado a cercare una reference a DeleteFileA, la trovo alla riga:
004032F7 FF1558634000 call [00406358]
bene, come vediamo il compilatore ha risolto la import con l'indirizzo 00403658. Leviamo l'image base e ci resta 00006358. Cosa c'è a 6358 nel file? Il FirstThunk di DeleteFileA. Ora l'ultimo passo: andiamo in sice e settiamo un bel bpx DeleteFileA. Provate ad aprire un file di testo con il notepad, e il sice popperà nel modulo KERNEL32 all'indirizzo BFFA19E5. Ecco quindi che abbiamo raggiunto il codice della funzione DeleteFileA.
Visto che anche l'IT è facile ora?
-- Recourse --
Pare che abbiano sbagliato la sintassi sul PEditor: questo è il campo Resource. Nelle risorse troviamo le stringhe, le bitmap, le finestre, etc. Per il notepad abbiamo l'rva 00007000 ed il size di 51BC.
Non riporto tutti i bytes, ma ci andremo solo ad occupare della struttura directory di questo campo e vedremo un pò come funzia. A 7000 troviamo la directory:

RESOURCE FLAGS	00 00 00 00
TIME/DATE STAMP	11 97 56 01
MAJOR VERSION | MINOR VERSION	04 00 00 00
# NAME ENTRY | # ID ENTRY	00 00 07 00
RESOURCE DIR ENTRIES	03 00 00 00

()- Resource Flags
Non usato, sta sempre a zero.
()- Time/Date Stamp
Dword che contiene data e creazione delle risorse.
()- Major/Minor Version
Due dwords, nel nostro caso Major = 4 Minor = 0. Semplicemente il numero della versione.
()- #Name Entry/ #ID entry
Name entry contiene il numero di oggetti all'inizio dell'array di Directory che hanno i nomi associati.
ID entry contiene il numero di oggetti che hanno delle dwords come nome associato.
Nel nostro caso Name Entry è pari a 0, e ID entry è 7. Detto così non è molto chiaro, spiegheremo tutto meglio fra poco.
()- Resource Dir Entries
In seguito alla directory c'è un array variabile di elementi che fanno parte della directory entries.
Ora torniamo a Name/ID entry: quelle due words ci dannò il numero di oggetti che hanno o una stringa o un numero come nome identificatore. Nel nostro caso vediamo che abbiamo sette oggetti che sono identificabili tramite interi a 32 bit. La struttura di un direcotry entry è fatta nel modo seguente:
NAME RVA/INTEGER ID DWORD
E | DATA ENTRY RVA/SUBDIR RVA DWORD
dove la prima dword è una union tra i campi NAME o INTEGER: può contenere o l'rva che punta al nome della risorsa (rva a 31 bit però) o un intero a 32 bit che indica l'identificativo della risorsa.
La seconda dword invece è determinata dal primo bit più a sinistra (E):
se E = 0 stiamo considerando i dati di una risorsa
e il seguente campo a 31 bit ci indica l'rva che punta ai dati
se E = 1 stiamo considerando una sottodirectory
e il seguente campo ci indica l'rva del Subdirectory Entry.
bene, ora per capire quello detto facciamo il classico esempio. Prendiamo le risorse del notepad. Queste cominciano all'indirizzo 7000. Ci andiamo e vediamo i seguenti bytes:
00 00 00 00 - 11 97 56 01 - 04 00 00 00 - 00 00 07 00
questi fanno parte della directory e identificano i campi Resource Flags, TimeDate stamp, major/minor version, Name/ID entry, e questo lo abbiamo visto. Dopo queste dwords inizia l'array di strutture Resource Dir Entries (siccome ID entry è sette, prendiamo sette strutture da 2 dwords):

03 00 00 00 - 48 00 00 80 - 04 00 00 00 - 98 00 00 80
05 00 00 00 - B0 00 00 80 - 06 00 00 00 - D0 00 00 80
09 00 00 00 - 00 01 00 80 - 0E 00 00 00 - 20 01 00 80
10 00 00 00 - 40 01 00 80

queste strutture vanno considerate a gruppi di 2 dwords. Vediamone il significato:

00000003 numero identificativo
80000048 subdirectory all'offset 48

00000004 numero identificativo
80000098 subdirectory all'offset 98

00000005 numero identificativo
800000B0 subdirectory all'offset B0
etc

dopo questa struttura troviamo un altra directory:

00 00 00 00 11 97 56 01 04 00 00 00 00 00 08 00

FLAGS TIME/DATE MAJ/MIN ENTRIES
abbiamo 8 entries, quindi prendiamo le successive otto strutture:

01 00 00 00 - 58 01 00 80
02 00 00 00 - 70 01 00 80 - 03 00 00 00 - 88 01 00 80
04 00 00 00 - A0 01 00 80 - 05 00 00 00 - B8 01 00 80
06 00 00 00 - D0 01 00 80 - 07 00 00 00 - E8 01 00 80
08 00 00 00 - 00 02 00 80

anche queste le interpretiamo nel modo precedente:

00000001 numero identificativo
80000158 subdirectory all'offset 158

00000002 numero identificativo
80000170 subdirectory all'offset 170
etc..

e dopo troviamo un altra directory, seguite da altre strutture Resource Dir Entries etc..
ora però abbiamo trovato solo subdirectory, ancora non sappiamo dove sono effettivamente i dati relativi alle risorse. Abbiamo trovato sempre strutture la cui seconda dword aveva il primo bit attivo (8xxxxxxx) quindi adesso cerchiamo qualche risorsa vera. La prima la trovo all'offset 7158:

00 00 00 00 - 11 97 56 01 - 04 00 00 00 - 00 00 01 00

ha una sola entry, quindi prendo la seguente struttura:

10 04 00 00 - 38 03 00 00
NUM ID DATA OFFS

stavolta la seconda dword ha il primo bit a 0 (0xxxxxxxx), quindi vuol dire che i rimanenti 31 bit ci indicano l'offset dei dati della risorsa: 00000338. Ora questo non è un indirizzo vero e proprio, ma è un valore da sommare al'rva della Resource(7000): 7000+0338 = 7338. Andiamo a vedere a questo offset:

78 74 00 00 - 28 01 00 00
E4 04 00 00 - 00 00 00 00 - A0 75 00 00 - E8 02 00 00
E4 04 00 00 - 00 00 00 00 - 88 78 00 00 - E8 02 00 00
E4 04 00 00 - 00 00 00 00 - 70 7B 00 00 - 28 01 00 00
E4 04 00 00 - 00 00 00 00 - 98 7C 00 00 - 68 06 00 00
E4 04 00 00 - 00 00 00 00 - 00 83 00 00 - A8 08 00 00
E4 04 00 00 - 00 00 00 00 - A8 8B 00 00 - A8 0E 00 00
E4 04 00 00 - 00 00 00 00 - 50 9A 00 00 - 68 05 00 00
E4 04 00 00 - 00 00 00 00 - B8 9F 00 00 - 06 03 00 00
E4 04 00 00 - 00 00 00 00 - 24 A3 00 00 - CE 00 00 00
E4 04 00 00 - 00 00 00 00 - F4 A3 00 00 - A6 04 00 00
E4 04 00 00 - 00 00 00 00 - 9C A8 00 00 - 64 02 00 00
E4 04 00 00 - 00 00 00 00 - 00 AB 00 00 - 32 0A 00 00
E4 04 00 00 - 00 00 00 00 - 34 B5 00 00 - A8 02 00 00
E4 04 00 00 - 00 00 00 00 - DC B7 00 00 - 38 05 00 00
E4 04 00 00 - 00 00 00 00 - 14 BD 00 00 - 48 00 00 00
E4 04 00 00 - 00 00 00 00 - 5C BD 00 00 - 68 00 00 00
E4 04 00 00 - 00 00 00 00 - C4 BD 00 00 - 22 00 00 00
E4 04 00 00 - 00 00 00 00 - E8 BD 00 00 - 5A 00 00 00
E4 04 00 00 - 00 00 00 00 - 44 BE 00 00 - 78 03 00 00
E4 04 00 00 - 00 00 00 00

queste sono tutte le strutture puntate dalle resource dir entries. Ognuna di queste strutture è formata da 4 dwords, che hanno il seguente significato:

DATA RVA
SIZE
CODEPAGE
RESERVED

quindi la prima struttura ha i seguenti dati:

00007478 DATA RVA
00000128 SIZE
000004E4 CODEPAGE
00000000 RESERVED

. Data rva
contiene l'rva che punta ai dati veri e propri (finalmente) della risorsa.
. Size
è la grandezza dei dati della risorsa

. CodePage
il valore che dovrebbe essere usato nella decodifica dei code point all'interno del blocco di dati.
. Reserved
riservato! Deve essere 0.

quindi sappiamo finalmente dove sono immagazzinati i dati della prima risorsa. Andiamo all'offset 7478 e troviamo il blocco:

28 00 00 00 10 00 00 00
20 00 00 00 01 00 04 00 00 00 00 00 80 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 80 00 00 80 00 00 00 80 80 00
80 00 00 00 80 00 80 00 80 80 00 00 C0 C0 C0 00
80 80 80 00 00 00 FF 00 00 FF 00 00 00 FF FF 00
FF 00 00 00 FF 00 FF 00 FF FF 00 00 FF FF FF 00
00 00 00 00 00 00 00 00 00 00 07 77 77 77 77 77 80
00 00 07 77 77 77 77 70 00 00 07 77 77 77 FF 70
00 00 07 77 77 77 77 70 00 00 00 00 00 77 77 70
6E EE EE EE E6 07 7F 70 0E EE EE EE EE 07 77 70
06 EE EE EE EE 60 77 70 00 EE EE EE EE E0 77 70
00 6E EE EE EE E6 07 70 00 0E E6 66 66 EE 07 70
00 06 E6 66 66 6E 60 70 00 00 EE EE EE EE E0 70
00 00 08 66 86 68 60 00 00 00 00 00 00 00 00 00
F8 01 00 00 F0 00 00 00 F0 00 00 00 F0 00 00 00
F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80 00 00 00 80 00 00 00 C0 00 00 00 C0 00 00 00
E0 00 00 00 E0 00 00 00 F0 01 00 00 F8 03 00 00
28 00 00 00 20 00 00 00

Questo corrisponde al disegnino dell'icona del notepad. Basta cambiare questi bytes per cambiare l'icona del notepad (vedremo fra un pò il visualizzatore di risorse). Si procede allo stesso modo per tutte le altre risorse.

-- Exception -- & -- Security --

Questi due campi sono sconosciuti, non ho trovato nessuna documentazione.

-- Basereloce --

Sta per base relocation. Abbiamo già parlato delle rilocazioni, abbiamo detto che le variabili statiche, le stringhe etc venivano caricate ad un diverso indirizzo da quello specificato in ImageBase. Il campo basereloc ci dà un puntatore (alla sezione .reloc) alla struttura che contiene le informazioni per effettuare la rilocaione. Nel caso del notepad abbiamo un D000 e un size di 93C. Al solito andiamo a vedere con i nostri occhi la Relocation Table (anche chiamata Fixup Table):

00 10 00 00 - 68 02 00 00 - D5 30 1B 31 - 36 31 45 31
52 31 5A 31 - 61 31 67 31 - 6F 31 76 31 - 7D 31 81 31
8A 31 A2 31 - AF 31 BD 31 - C2 31 CC 31 - D1 31 D9 31
E7 31 EC 31 - F6 31 FB 31 - 06 32 0F 32 - 23 32 29 32
3D 32 74 32 - 7B 32 B9 32 - C3 32 C8 32 - DC 32 E6 32 continua....

ho riportato solo un pezzo, tanto basta e avanza. Al solito vediamo la struttura teorica di questa roba:

PAGE RVA
BLOCK SIZE
TYPE/OFFSET | TYPE/OFFSET
TYPE/OFFSET | continua...

la prima dword (page rva) è 00001000, ed ci dà l'rva che serve per individuare dove andare ad effettuare la rilocazione, poi BlockSize indica il numero di bytes del blocco da rilocare. Tutte le word successive sono TYPE/OFFSET formate nel modo:
bit 15---11---------0
Type Offset
in cui Type indica il tipo di rilocazione a seconda del suo valore:
. 0 = Assoluta
E' un nop. La rilocazione viene saltata.
. 1 = High
L'offset punta alla parte alta (16 bit a sinistra) di un indirizzo a 32 bit, in questa parte alta vanno applicati i 16 bit di rilocazione.
. 2 = Low
Come prima, l'offset punta alla parte bassa (16 bit a destra) di un indirizzo a 32 bit, ed è lì che va applicata la rilocazione.
. 3 = HighLow
L'intera relocazione a 32 bit va applicata all'intero indirizzo a 32 bit.
. 4 = HighAdjust
Si tratta sempre di una rilocazione a 32 bit, ma con dei calcoli in più (che sono incomprensibili).
. 5 = MipsJmpAddr
sconosciuto
. 6 = Section

sconosciuto
. 7 = Rel32
sconosciuto
tornando all'esempio pratico, nella nostra Relocation Table avevamo le seguenti info:
00 10 00 00 - 68 02 00 00 - D5 30 1B 31 - 36 31 45 31 etc...
cioè la rilocazione inizierà all'rva 00001000 e sarà lunga 0268h byte. La lunghezza però comprende anche le 2 dword iniziali, quindi per sapere quante rilocazioni occorreranno dobbiamo sottrarre 8 bytes a size (gli 8 bytes delle prime 2 dwords), 268h - 8 = 260h, e ora dividiamo per 2 il numero di bytes (così otteniamo le words delle reloc), 260h / 2 = 130h = 304 (dec) rilocazioni.
-- Debug --
Anche questo campo ci dà l'rva di una struttura. Questa contiene info per il debugging, la descriverò fra poco.
-- Copyright -- & -- Globalptr --
Sconosciuti
-- Tls Table --
Ci dà l'indirizzo del Thread Local Storage. Ne parleremo fra poco.
-- Load Config --
sconosciuto
-- Bound Import --
Questo per rispondere a WebCide che mi ha fracassato di domande l'altro giorno in chat :-P. Abbiamo già visto gli OriginalFirstThunk e i FirstThunk. Quando il PE loader carica un eseguibile, trova le imports, carica in OriginalFirstThunk gli rva del file ai nomi di funzioni, al tempo stesso li carica anche nella struttura dei FirstThunk. Poi carica le dll varie, ottiene gli entry point delle funzioni da importare e le va a sostituire ai valori dei FirstThunk. Ora, nella maggior parte dei casi l'entry point di una funzione in una dll sarà sempre lo stesso, quindi possiamo mettere nei FirstThunk direttamente gli entry point, in modo che il loader non perda tempo a metterceli per noi. Questo processo si chiama "binding". Ovviamente se abbiamo i FirstThunk già settati sugli Entry point delle funzioni, ma per un qualsiasi motivo a quegli indirizzi non vengono trovate le funzioni, il loader se ne accorge (o dal numero di versione della dll, o da una avvenuta relocazione) e quindi ecco che ritorna agli OriginalFirstThunk per andarsi a ricalcolare l'entry point giusto delle funzioni considerate.
-- Import Address Table --
Abbiamo già trattato la IAT nella sezione della IT, cmq questo valore è superfluo, in quanto già specificato nella directory entry dell'IT. Indica il puntatore rva all'inizio della IAT.
Okkey, adesso rimangono solo i pulsantini del gruppo view:
Exports, Imports, Resource, Debug, TLS.
Ognuno di essi apre una finestra a parte, e servono solo a visualizzare in una interfaccia utente i dati delle export table, import table, risorse etc.
** EXPORTS **
Prendo la solita wz32.dll e guardo le imports. Sulla sinistra sono riportati tutti i valori della struttura che già abbiamo commentato, nella destra avete le funzioni ordinate per indice, con i relativi rva e nome.
** IMPORTS **
Nel riquadro superiore avete i nomi dei moduli importati (i file dll), cliccate su un modulo e nel riquadro inferiore appariranno le funzioni importate da quel modulo, complete di rva, offset, hint e nome (che già abbiamo descritto in dettaglio).
Il riquadro superiore merita un attimino di attenzione: nella descrizione dei moduli compaiono le voci corrispondendi a FirstThunk, OriginalFirstThunk etc. che ho già descritto.
** RESOURCES **
Beh, avete il visualizzatore ad albero delle risorse, avete anche il visualizzatore delle icone! Mitico!
A destra avete le info sui valori che ho già discusso prima (name entries, etc..), e per ogni risorsa avete in basso a destra la locazione (rva) e il suo size. La prima risorsa è un icona, ed è la classica icona del notepad (come già visto prima; se osservate attentamente il blocco ascii relativo all'icona, vi sembrerà di intravedere il disegnino del notepad! infatti è così).
** DEBUG **
Per vedere qualcosa in questo campo dovete creare e compilare un eseguibile con le info di debug (il visual c++ lo fa in automatico). Debug quindi vi aprirà un dialog con le informazioni delle strutture di debug. Una struttura di debug è formata nel seguente modo:

CHARACTERISTICS
TIME/DATE STAMP
MAJOR VERSION | MINOR VERSION
DEBUG TYPE
DATA SIZE
DATA RVA
DATA SEEK

dove: Characteristics è sempre a zero,
TimeDate Stamp è la data e ora della creazione delle info di debug
Maj/Min Version è la versione delle info di debug
Debug Type è il formato delle info di debug, che può essere in simboli COFF, FPO, o CodeView
Data size è il numero di bytes dei dati di debug (escluse le debug directory)
Data rva è appunto l'rva dei dati di debug
Data seek è il seek value dall'inizio del file fino ai dati di debug
Il campo Debug (nella Directory) ci indica l'rva dell'array di queste strutture di debug. Il debug viewer semplicemente ci permette di scorrere una ad una tutte queste strutture. Francamente non sembra

funzionare molto bene.
** TLS **
Acronimo di Thread Local Storage. Questa struttura non la descriverò, anche perchè il suo funzionamento non mi è ancora molto chiaro, e ciò è dovuto al fatto che non riesco a trovare un eseguibile che abbia una TLS. La TLS contiene dei dati che verranno usati dai thread. In particolare la struttura contiene anche un array di rva dei Callback, cioè un array di funzioni che devono essere richiamate prima e dopo la creazione del thread.
---+
-o NOTE
---+
Nella descrizione della IT ho usato il modello classico, che è quello della struttura:

Image_Import_Descriptor Struct
union
characteristics dd? (Import Flags)
OriginalFirstThunk dd ?
ends
TimeDateStamp dd ?
ForwarderChain dd ?
Name1 dd ?
FirstThunk dd ?
Image_Import_Descriptor Ends

però nella documentazione del pe.txt rilasciata da Microschif troviamo per l'IT la seguente struttura:

IMPORT FLAGS
TIME/DATE STAMP
MAJOR VERSION | MINOR VERSION
NAME RVA
IMPORT LOOKUP TABLE RVA
IMPORT ADDRESS TABLE RVA

la differenza è data dal fatto che alcuni compilatori usano la seconda struttura per costruire la IT.
Con questa struttura non avete gli OriginalFirstThunk, e dovete risalire al nome delle funzioni usando i FirstThunk. Poi l'array di FirstThunk viene sovrascritto con gli entry point delle funzioni importate.
E' chiaro che in questo caso non potete avere le bound import. Di solito possiamo capire quale struttura ci troviamo di fronte semplicemente dalla prima dword: se la struttura rientra nel primo caso, allora la prima dword conterrà l'rva per i nomi delle funzioni, se la struttura rientra nel secondo caso allora la prima dword sarà settata a zero. Praticamente è proprio questa union che determina il tipo di struttra della IT.
Quando ho parlato del Rebuilder ho detto che potreste ritrovarvi ad aver ricostruito la IT di un prog, però scoprirete che tale prog funziona solo sul vostro pc. Il problema è legato ai Thunks. Infatti nei FirstThunk troviamo o gli indirizzi degli entry point delle funzioni importate, oppure troviamo una copia degli OriginalFirstThunk. In tal caso il loader pensa ad ottenere gli entry point delle funzioni e a sovrascriverle alla copia degli OriginalFirstThunk per ottenere i veri FirstThunk. Se voi dumperete la IT dopo questo processo, vi ritroverete una IT che non è l'originale, ma il file funziona in quanto ha già prepatchati gli entry point delle funzioni. Può capitare che non tutte le dll carichino il loro codice sempre allo stesso indirizzo (per vari motivi). Nello stesso pc al 99% delle volte gli indirizzi sono sempre quelli, ecco perchè sul vostro pc funziona, però su un altro pc gli indirizzi probabilmente saranno diversi, e quindi il file non funzionerà. Voi direte: ma abbiamo sempre gli OriginalFirstThunks, si ma questo se il file usa la prima struttura della IT. Se usa la seconda descritta poc'anzi, allora non avremo gli OriginalFirstThunk e i FirstThunk sono sbagliati. Per risolvere il problema dovrete quindi dumpare la vostra IT prima che venga sovrascritta con i FirstThunk.
---+
-o LIBRARY, THANKS, LINKS AND E-MAIL
---+
Per documentarvi meglio sul formato PE vi consiglio l'utilissimo documento PE.TXT che trovate un pò dovunque (quello rilasciato da Microzozz Developers Support) e anche la documentazione di LuevelsMeyer.
Inoltre vi consiglio i tutorial sul PE che trovate sul RingZero, e vari tutorial sul Manual Unpacking che trovate alla UIC. Vi consiglio anche i testi di Iczelion e tanti altri buoni testi che adesso non mi vengono in mente, ma ce ne sono molti. Studiate, studiate!
Saluto tutta la UIC, il RingZero e gli Spippolatori, un salutone a Yado che col suo Krypton ci costringe a studiare molto sul packing! Non dimentichiamo poi gli HackManiaci che mi hanno coinvolto nello scrivere questo documento. And last but not least un salutone a Syscalo e Ritz che hanno supervisionato questo testo (e che si sono incazzati perchè non li avevo salutati :-P)
E poi non poteva mancare il saluto al Quaqquaro!!! Ma io e te non dovevamo fare a botte? hihihihihi :-P
Some linkz:
http://ringzer0.goldrake.com - il ringzero! Grazie di esistere!
www.hackmaniaci.cjb.net - bravi guaglioni
www.uic-spippolatori.com - la UIC (università italiana crackers)
www.spippolatolri.com - gli spippolatori
...and MY stuff:
www.andreageddon.8m.com - il mio sito
andreageddon@hotmail.com - la mia email
irc.azzurra.it -> #crack-it #hackmaniaci - canali bazzicati su irc.

- EOF - (pant pant!)
AndreaGeddon

Note Finali

Spero di essere stato abbastanza chiaro nelle spiegazioni. Se trovate errori o qualsiasi cosa che non vi garba, segnalatemelo.
Ciauuuz.
AndreaGeddon

Disclaimer

Queste informazioni sono solo a scopo puramente didattico.