Nebojme se upravovat PE soubory I.

Vložil/a RubberDuck [1], 28 Srpen, 2013 - 00:50

- Cracking [2]
- Programming [3]
- Virus & Worms [4]

Cílem následujícího seriálu je popsat techniky úpravy PE kódu. V prvním díle se seznámíme s nezbytnou strukturou PE souboru a popíšeme si, jak vytvořit v souboru novou sekci pomocí editoru, a jak zajistit vykonání našeho kódu uloženého v této sekci.

Úvod
Práce s jakýmkoliv souborovým formátem můžeme považovat za vskutku delikátní záležitost. Pokud ne z pocitů moci nad souborem, které to člověku přináší, pak alespoň pro pocit jedinečnosti, protože se nejedná o příliš oblíbenou činnost. Prakticky se dá říct, že tuto činnost ovládají skutečně jen specialisté. Mezi tuto skupinu lidí patří autoři původní (čti infikujících) virů, AVeři, reverzní inženýři a programátoři pracující na kompilerech pro Windows.

Co potřebuji?
Tato otázka napadla snad každého, kdo tento článek otevřel. Většina lidí v této činnosti temnou magii. Tak temnou, že nepředpokládají, že by byli schopni se ji naučit. Asi vás zklamu ale ve skutečnosti se jedná o velmi přesný postup daný strukturou PE souboru s využitím jednoduché matematiky (sčítání, odčítání, násobení, dělení, zaokrouhlování) a editor, který umí pracovat s PE soubory. To je vše. Víc aktuálně není třeba.

Co je to PE?
PE je zkratka pro Portable Executable. Jedná se o binární strukturu souboru sloužící na operačních systémech Windows jako spustitelné programy (EXE, DLL atd.). Obecně se skládá ze dvou částí: hlaviček a surových dat. Tuto strukturu velmi dobře popsal Matt Pietrek ve svých dvou dokumentech[1][2] o PE souborech. Jejich přečtení velmi doporučuji každému, kdo chce pohlédnout hlouběji do této problematiky. Tento seriál se totiž bude zabývat pouze nezbytnými částmi PE souboru.

Struktura PE
Jak bylo popsáno výše, PE se skládá z dvou hlavních částí: hlaviček a surových dat. Hlavičky mají za cíl pomoci loaderu (zavaděči programu) se lépe zorientovat v surových datech PE souboru. Obecně se hlavičky dělí na dvě hlavní části: DOS hlavičku a PE hlavičku.

DOS hlavička
DOS hlavička je určitý přežitek a slouží pouze pro zpětnou kompatibilitu s operačním systémem DOS. Obecně se pro ni používá označení IMAGE_DOS_HEADER[3] a má následující strukturu:

```
typedef struct _IMAGE_DOS_HEADER
{
     WORD e_magic;
     WORD e_cblp;
     WORD e_cp;
     WORD e_crlc;
     WORD e_cparhdr;
     WORD e_minalloc;
     WORD e_maxalloc;
     WORD e_ss;
     WORD e_sp;
```

```
    WORD e_csum;
    WORD e_ip;
    WORD e_cs;
    WORD e_lfarlc;
    WORD e_ovno;
    WORD e_res[4];
    WORD e_oemid;
    WORD e_oeminfo;
    WORD e_res2[10];
    LONG e_lfanew;
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

Z celé této struktury jsou pro nás důležité pouze pole:
- e_magic (signatura s konstantní hodnotou 'MZ' - tato hodnota značí začátek struktury IMAGE_DOS_HEADER a jedná se o jednu ze dvou signatur, které loader kontroluje při zavádění binárního souboru do systému)
- e_lfanew (slouží jako offset PE hlavičky)
Nyní je na čase vysvětlit si, co je to offset. Offset je logická adresa, jejímž přičtením k bázové adrese získáme teprve fyzickou adresu, kde se data reálně nachází. Pokud je například offset 0x1234 a bázová adresa je 0x10000000, pak je fyzická adresa 0x10001234. V našem případě přičteme offset k adrese počátku struktury IMAGE_DOS_HEADER, což je pro nás bázová adresa a získáme adresu PE hlavičky.

PE hlavička
- PE hlavička obsahuje data spojená se strukturami PE souboru. Obecně se pro ni používá označení IMAGE_NT_HEADERS[4] a má následující strukturu:

```
typedef struct _IMAGE_NT_HEADERS {
  DWORD                 Signature;
  IMAGE_FILE_HEADER     FileHeader;
  IMAGE_OPTIONAL_HEADER OptionalHeader;
} IMAGE_NT_HEADERS, *PIMAGE_NT_HEADERS;
```

Pokud někoho zaskočila velikost této struktury, upřesním, že se jedná o pseudostrukturu. Ve skutečnosti se totiž jedná o strukturu signatury a dvou dalších struktur. PE signatura má konstantní hodnotu 'PE\0\0' a slouží jako značení začátku PE struktury.
Struktura IMAGE_FILE_HEADER[5] reprezentuje tzv. COFF formát. Abych se vyhnul dlouhému vysvětlování, představme si tuto strukturu jako popisovač základních vlastností PE souboru (což ve skutečnosti opravdu je). Struktura má následující formát:

```
typedef struct _IMAGE_FILE_HEADER {
  WORD  Machine;
  WORD  NumberOfSections;
  DWORD TimeDateStamp;
  DWORD PointerToSymbolTable;
  DWORD NumberOfSymbols;
  WORD  SizeOfOptionalHeader;
  WORD  Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER
```

Zde můžeme považovat za důležité pole Machine udávající architekturu, pro kterou je PE soubor určen (pro nás budou zajímavé pouze PE soubory s hodnotou IMAGE_FILE_MACHINE_I386, tedy x86, protože struktura x86 a x64 se od sebe v některých ohledech liší). Dále je pro nás důležité pole NumberOfSections. V tomto poli je uložena informace, kolik sekcí PE soubor používá. O sekcích bude řeč později. Obecně se pojmem sekce označuje část PE souboru uchovávající typově stejná data, například definované/nedefinované proměnné, kód programu atd.

Struktura IMAGE_OPTIONAL_HEADER[6] uchovává doplňkové informace o PE souboru. Ve skutečnosti se jedná o poměrně důležité informace. Pokud bychom se bez struktury IMAGE_FILE_HEADER obešli (v ideálním případě samozřejmě), bez struktury IMAGE_OPTIONAL_HEADER už by se jednalo o neřešitelný problém. Má následující strukturu:

```
typedef struct _IMAGE_OPTIONAL_HEADER {
  WORD                 Magic;
  BYTE                 MajorLinkerVersion;
  BYTE                 MinorLinkerVersion;
  DWORD                SizeOfCode;
  DWORD                SizeOfInitializedData;
  DWORD                SizeOfUninitializedData;
  DWORD                AddressOfEntryPoint;
  DWORD                BaseOfCode;
  DWORD                BaseOfData;
  DWORD                ImageBase;
  DWORD                SectionAlignment;
  DWORD                FileAlignment;
  WORD                 MajorOperatingSystemVersion;
  WORD                 MinorOperatingSystemVersion;
  WORD                 MajorImageVersion;
  WORD                 MinorImageVersion;
  WORD                 MajorSubsystemVersion;
  WORD                 MinorSubsystemVersion;
  DWORD                Win32VersionValue;
  DWORD                SizeOfImage;
  DWORD                SizeOfHeaders;
  DWORD                CheckSum;
  WORD                 Subsystem;
  WORD                 DllCharacteristics;
  DWORD                SizeOfStackReserve;
  DWORD                SizeOfStackCommit;
  DWORD                SizeOfHeapReserve;
  DWORD                SizeOfHeapCommit;
  DWORD                LoaderFlags;
  DWORD                NumberOfRvaAndSizes;
  IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER;
```

Z celé struktury nás budou zajímat následující pole:
- SizeOfCode (celková velikost sekce kódu, pokud je těchto sekcí víc, pak součet všech těchto sekcí v bajtech)
- SizeOfInitializedData (celková velikost inicializovaných data sekcí, pokud je těchto sekcí víc, pak součet všech těchto sekcí v bajtech)
- SizeOfUninitializedData (celková velikost neinicializovaných data sekcí, pokud je těchto sekcí víc, pak součet všech těchto sekcí v bajtech)
- AddressOfEntryPoint (adresa vstupního bodu aplikace - na této adresa začíná vykonávání programu)
- ImageBase (bázová adresa celého obrazu, zjednodušeně řečeno - jedná se o adresu v paměti, na kterou je vždy při zavádění binárka namapována)
- SectionAlignment (zarovnání dat v paměti)
- FileAlignment (zarovnání surových dat na disku)
- SizeOfImage (celková velikost obrazu zaokrouhlená na násobek SectionAlignment)
- SizeOfHeaders (celková velikost všech hlaviček zaokrouhlená na hodnotu FileAlignment)

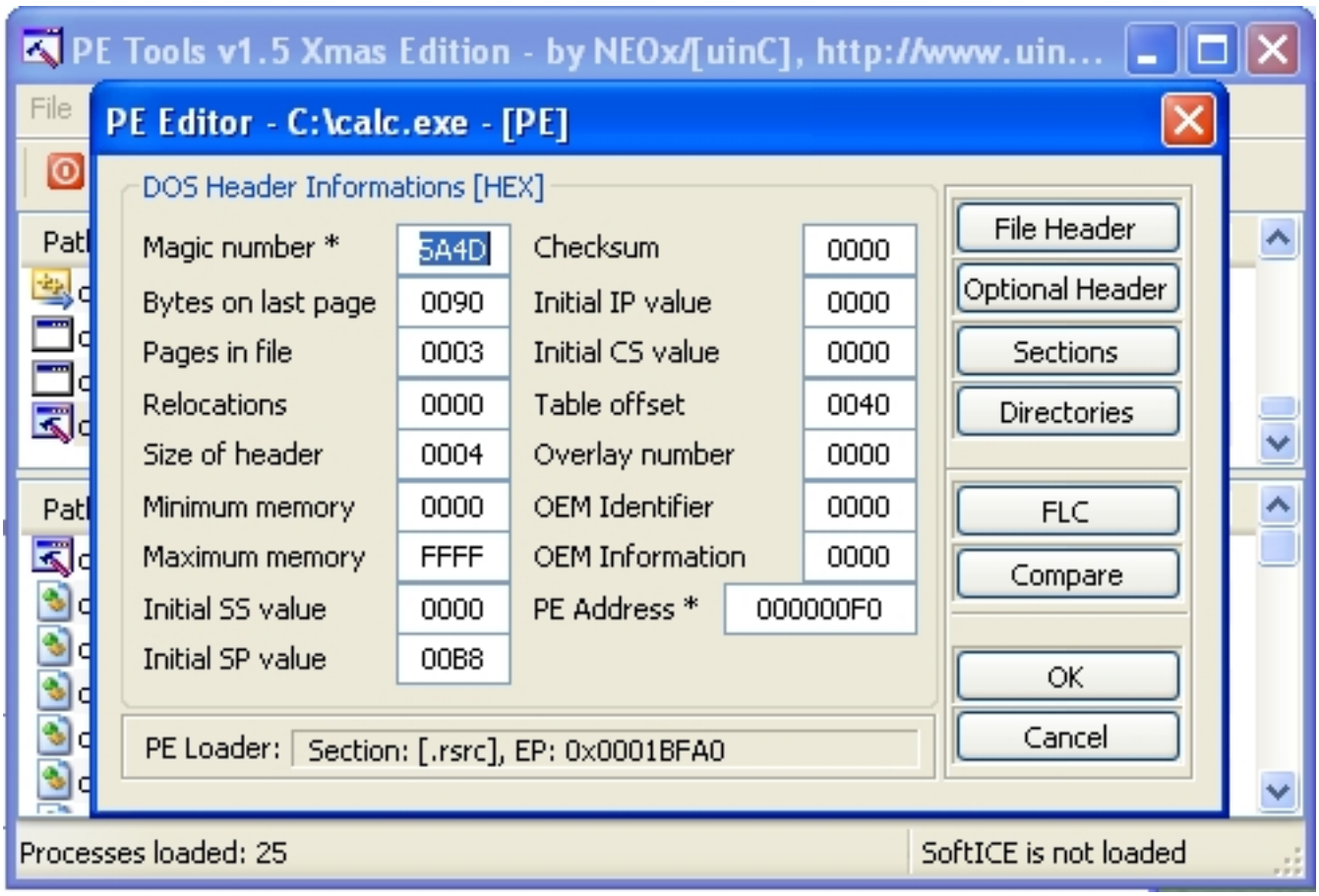
Tímto jsme se dostali na konec hlavičky PE, ale ne na konec PE souboru. Za PE hlavičkou se nachází hlavičky pro jednotlivé sekce PE souboru. Jejich počet je dán hodnotou NumberOfSections ve

struktuře IMAGE_FILE_HEADER (vzpomínáte? ;)) a jsou řazeny za sebou, takže je můžeme sekvenčně procházet. Každá hlavička sekce je definována strukturou IMAGE_SECTION_HEADER[7] a má následující strukturu:

```
typedef struct _IMAGE_SECTION_HEADER {
  BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
  union {
    DWORD PhysicalAddress;
    DWORD VirtualSize;
  } Misc;
  DWORD VirtualAddress;
  DWORD SizeOfRawData;
  DWORD PointerToRawData;
  DWORD PointerToRelocations;
  DWORD PointerToLinenumbers;
  WORD  NumberOfRelocations;
  WORD  NumberOfLinenumbers;
  DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

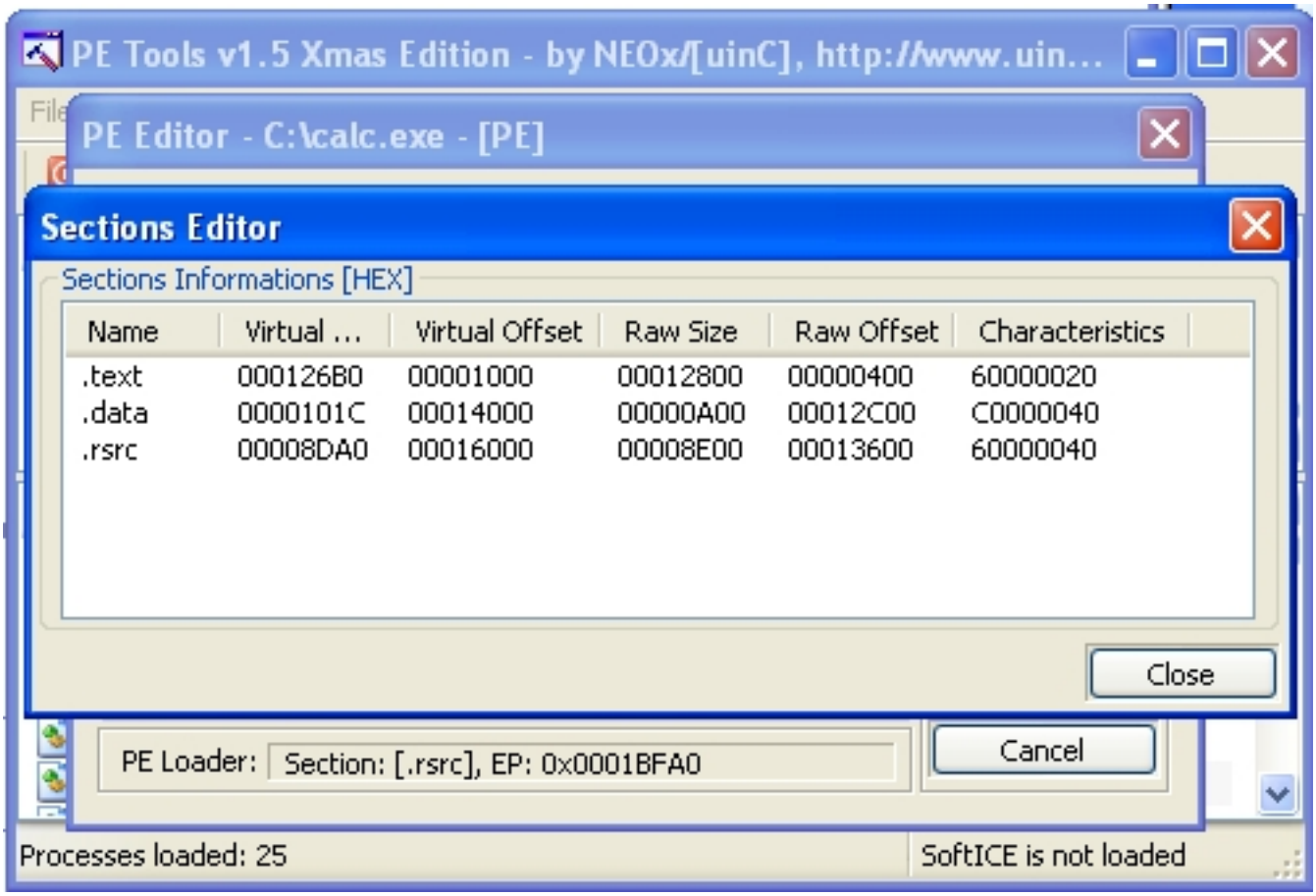
Nyní přejdeme ke struktuře IMAGE_SECTION_HEADER. Ta je pro tento díl velmi důležitá. Struktura IMAGE_SECTION_HEADER popisuje jednotlivé sekce PE souboru. Sekce je blok dat, který spolu nějak souvisí. Například sekce kódu obsahuje vykonavatelný kód aplikace, sekce neinicializovaných dat obsahuje neinicializovaná data a proměnné. Každá sekce má svou hlavičku. V hlavičce je uvedeno, kde daná sekce začíná, jak je veliká a co je možné s daty v dané sekci provádět (zda je možné je číst, zapisovat, vykonávat nebo kombinace předchozích možností). Každá sekce má dvě startovací adresy. Jednu pro data na disku, druhou pro data v paměti. Důvodem, proč jsou tyto hodnoty většinou různé je způsob, jak se v jednotlivých případech data zaokrouhlují (tzv. zarovnávají) (ve skutečnosti se jedná hlavně o to, jak samotný operační systém pracuje s adresami). V případě dat v paměti je použita hodnota SectionAlignment. Ta standardně bývá 1000h. U dat na disku se používá hodnota FileAlignment. Zde se nejčastěji setkáváme s hodnotou 200h. Při zarovnávání je velikost sekce/dat doplněna nulami až do násobku zarovnání (v některých zvláštních případech se místo nul používají jiné hodnoty, ale o tom jindy). Ve struktuře IMAGE_SECTION_HEADER jsou pro nás nejdůležitější pole:
- Name (udává jméno sekce - maximální délka jména sekce je 8 bajtů a při délce 8 bajtů nepoužívá ukončovací NULL bajt; standardně začíná tečkou)
- VirtualSize (udává velikost sekce v paměti - může, ale nemusí bát násobek SectionAlignment)
- VirtualAddress (udává startovací adresu sekce v paměti - musí být násobkem SectionAlignment)
- SizeOfRawData (udává velikost sekce na disku - musí být násobkem FileAlignment)
- PointerToRawData (udává startovací adresu sekce na disku - musí být násobek FileAlignment)

Nová sekce v PE pomocí editoru
Tím bychom měli vyčerpány aktuálně požadované struktury a můžeme se vrhnout do bližšího popisu. Jako ukázkový příklad jsem vytvořil EXE soubor[8], jenž vypíše MessageBox a ukončí se pomocí ExitProcess. Cílem je přidat do EXE souboru vlastní sekci s vlastním kódem a přesměrovat na něj vykonávání hned po startu tak, aby nedošlo k porušení vykonávacího procesu.



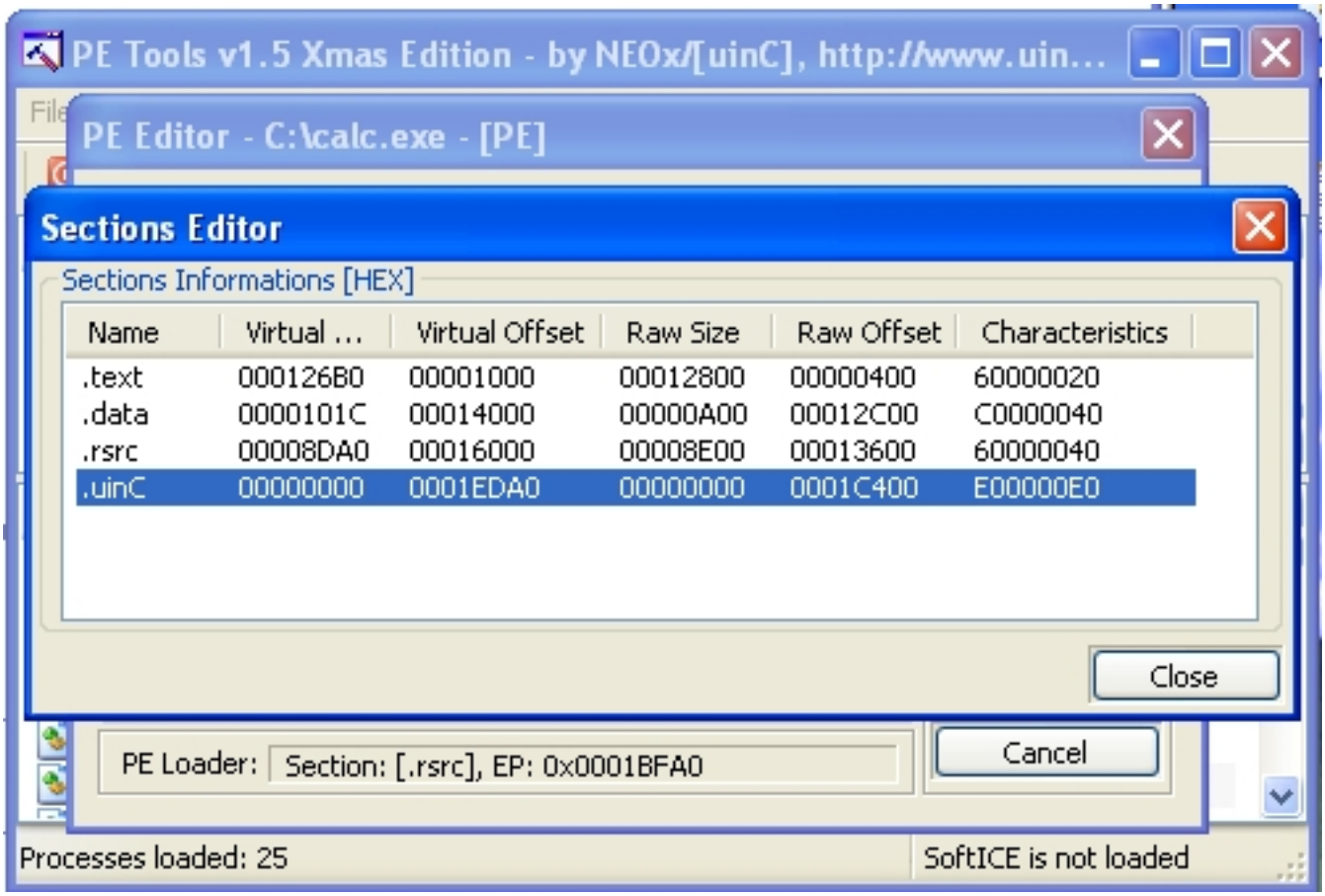
[5]
Otevřená aplikace messagebox.exe v PE Tools

Otevřeme si v aplikaci PE Tools[9] položku Sections, klikneme pravým tlačítkem myši a vybereme možnost Add Section. Na úplný konec seznamu všech sekcí se objevila jedna nová s názvem .uinC. Jak je vidět, PE Tools doplnil některé z hodnot za nás. Konkrétně PointerToRawData a VirtualAddress. Co tato pole znamenají jsme si vysvětlili výše. Nyní si řeknem, jak tyto hodnoty spočítat ručně:



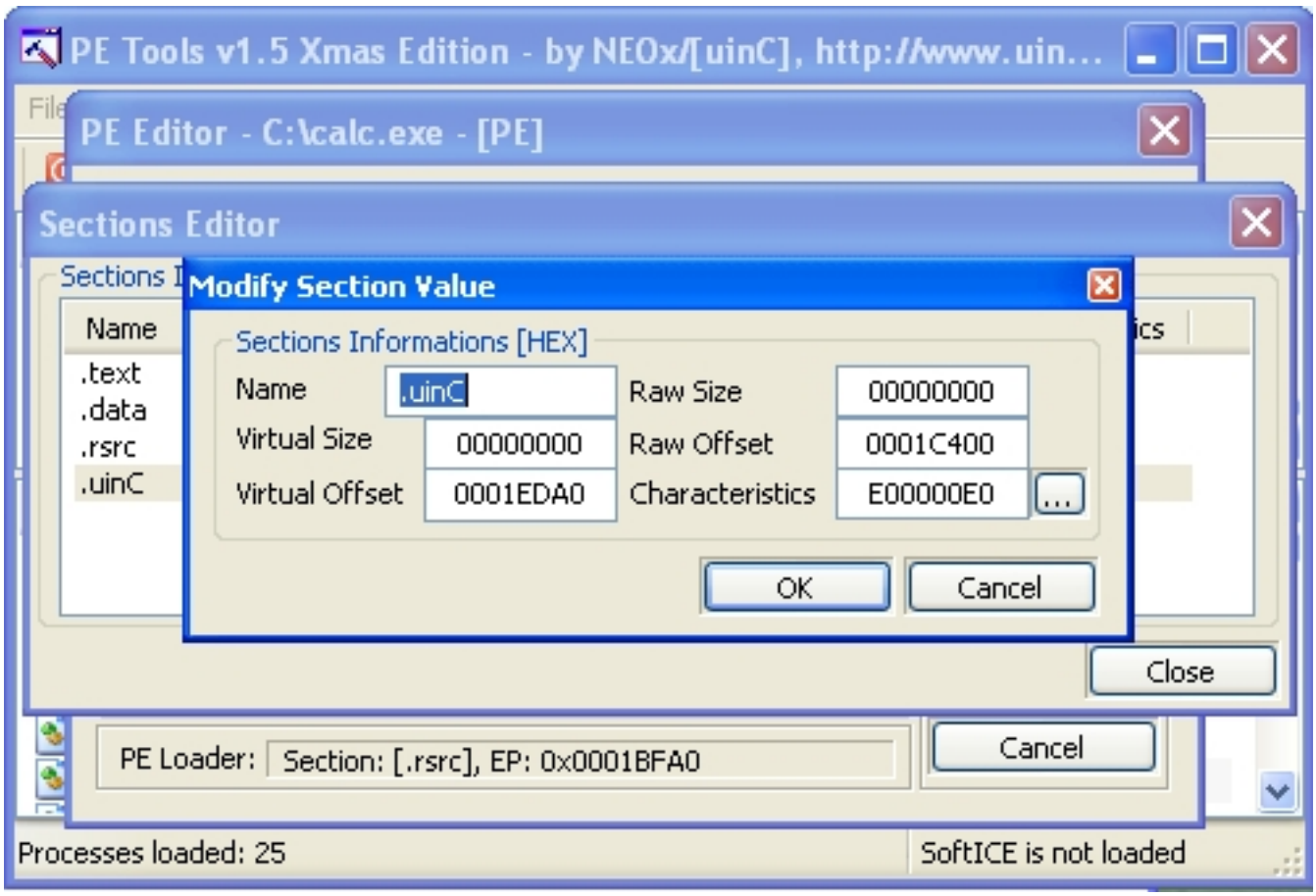
[6]
Zobrazené původní sekce aplikace messagebo.exe

PointerToRawAddress je roven součtu PointerToRawAddress předchozí sekce a SizeOfRawData předchozí sekce. Tato hodnota navíc musí být zarovnána na hodnotu FileAlignment. Zarovnáním se moc zdržovat nemusíme, protože hodnoty předchozí sekce již zarovnány jsou. VirtualAddress je rovna součtu VirtualAddress předchozí sekce a VirtualSize předchozí sekce zaokrouhlených na velikost SectionAlignment.



[7]
Nová sekce přidána do aplikace messagebox.exe

Zde již zaokrouhlení řešit musíme, protože velmi často hodnota VirtualSize není zarovnána na nejbližší násobek hodnoty SectionAlignment, takže by adresa další sekce nezačínala na korektní adrese. Nyní potřebujeme doplnit zbývající údaje. Konkrétně VirtualSize a SizeOfRawData. Začneme tím jednodušším. VirtualSize bude mít hodnotu velikosti přidávaného kódu. Takže pokud budu přidávat kód o velikosti 234 bajtů, bude tato hodnota EAh, což je 234d. V případě hodnoty SizeOfRawData musíme velikost přidávaného kódu ještě zarovnán na velikost FileAlignment. Dejme tomu, že FileAlignment má hodnotu 200h a velikost našeho kódu je 234 bajtů. Výsledná hodnota SizeOfRawData bude tedy 200h. Pokud by velikost našeho kódu byla 513 bajtů (201h), pak by byla hodnota SizeOfRawData rovna 400h. Poslední položkou je Characteristics. Vzhledem k tomu, že do sekce budeme přidávat vykonavatelný kód, musí mít sekce příznak minimálně executable. V opačném případě by při spuštění kódu došlo k chybě (výjimce) a pádu aplikace.



[8]
Úprava nové sekce v aplikaci messagebox.exe

Pro naše potřeby vložíme velmi krátký testovací kód. Vlastně se bude jednat pouze o přiřazení původní adresy vstupního bodu do registru EAX a následný skok na adresu uloženou v registru EAX. Adresu původního vstupního bodu aplikace najdeme v IMAGE_OPTIONAL_HEADER v poli AddressOfEntryPoint. K této hodnotě přičteme hodnotu uloženou v poli ImageBase a získáme tak absolutní adresu původního vstupního bodu. V případě naší aplikace je výpočet následující:

1130h + 400000h = 401130h

Tuto hodnotu uložíme do registru eax a odskočíme na danou adresu:

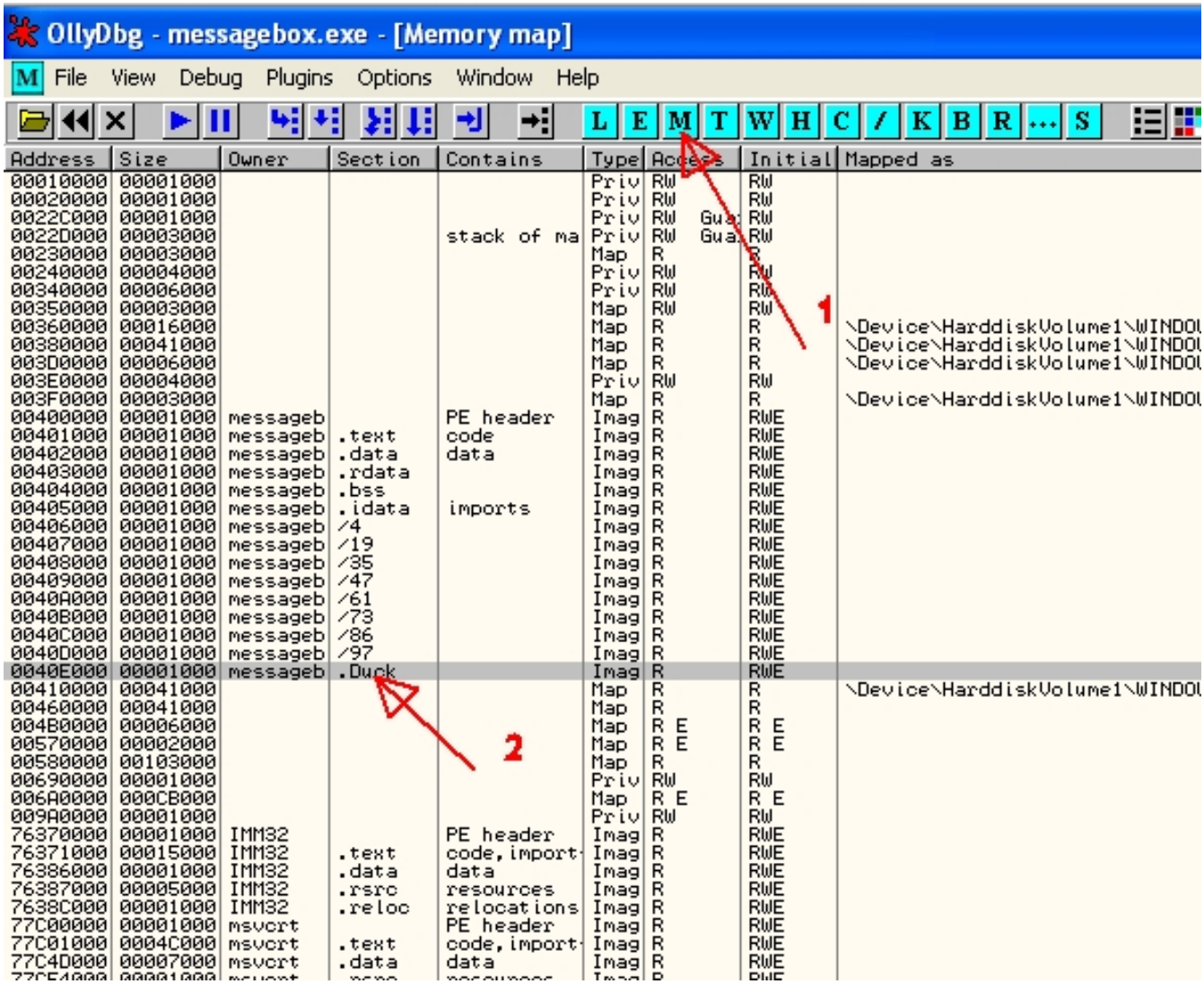
```
mov eax, 401130h
jmp eax
```

Ve formě opkódů vypadá výše uvedený kód následovně:

```
\xB8\x30\x11\x40\x00
\xFF\xE0
```

Z tohoto zápisu můžeme usuzovat, že náš kód má délku 7 bajtů. Hodnota VirtualSize tedy bude 7 a hodnota SizeOfRawData bude 200h (vzpomínáte na zaokrouhlení? ;)). Vložíme naše hodnoty do příslušných polí. Pokud má někdo zájem, může si přejmenovat sekci v poli Name. Ale nezapomeňte, že jméno může mít maximálně 8 bajtů! Já zvolím jméno .Duck :) A klikneme na OK. Tím jsme uložili do PE souboru naší hlavičku. Teď musíme upravit hodnotu SizeOfImage, SizeOfHeaders a AddressOfEntryPoint ve struktuře IMAGE_OPTIONAL_HEADER. S velikostí obrazu nám pomůže aplikace. Stačí kliknout a nová hodnota SizeOfImage je vypočítána. Jak vidíte, přičetla se přesně velikost našeho kódu. Ale pozor! Některé kompilátory tuto hodnotu zaokrouhlují na velikost

SectionAlignment, což nám příliš nevadí, ale je dobré o tom vědět. Stejný stav je i u pole SizeOfHeaders. Jak tuto hodnotu spočítat jsme si vysvětlili výše. Rovněž jsme si řekli, že ji musíme zarovnat na velikost FileAlignment. Ve většině případů není třeba s touto hodnotou pohybovat. Nikdy na to však nespoléhejte a hodnotu přepočítejte. Hodnotu AddressOfEntryPoint změníme na hodnotu VirtualAddress poslední sekce. Důvod je zřejmý. Chceme, aby se vykonal nejdříve náš kód a až následně ten originální. Proto musí být vstupním bodem aplikace právě naše nová sekce. V našem případě bude mít AddressOfEntryPoint hodnotu E000h. Nyní máme možnost změnit hodnotu pole SizeOfCode, protože naše nová sekce je sekce kódu. Pokud hodnotu SizeOfCode nezměníme, nic se nestane. Systém ji využívá orientačně, aby věděl, jak velkou část paměti bude asi kód potřebovat. Stejný stav věcí je pro pole SizeOfInitializedData (pro sekce s inicializovanými daty) a SizeOfUninitializedData (pro sekce s neinicializovanými daty). S posledními dvěma jmenovanými si více budeme hrát v některém z následujících dílů. Poslední věc, jenž je třeba upravit, je pole NumberOfSections ve struktuře IMAGE_FILE_HEADER. Jak název napovídá, udává toto pole počet sekcí PE souboru. Protože jsme jednu sekci přidali, musíme přičíst k hodnotě NumberOfSections jedničku. To však za nás již udělala aplikace PE Tools :) Tím máme upraveny všechny potřebné hodnoty a můžeme se vrhnout na vložení našeho kódu. Buď můžeme využít služeb hexadecimálního editoru a do souboru si připravit výše zmíněný opkód, který následně v tabulce Sections můžeme po kliknutí pravým tlačítkem na naši sekci vložit nebo využijeme možností Olly[10] debuggeru a sekci přidáme z něj. Já jsem se rozhodl pro druhou možnost (v praxi vlastně budeme využívat hexadecimální editor integrovaný do OllyDbg).

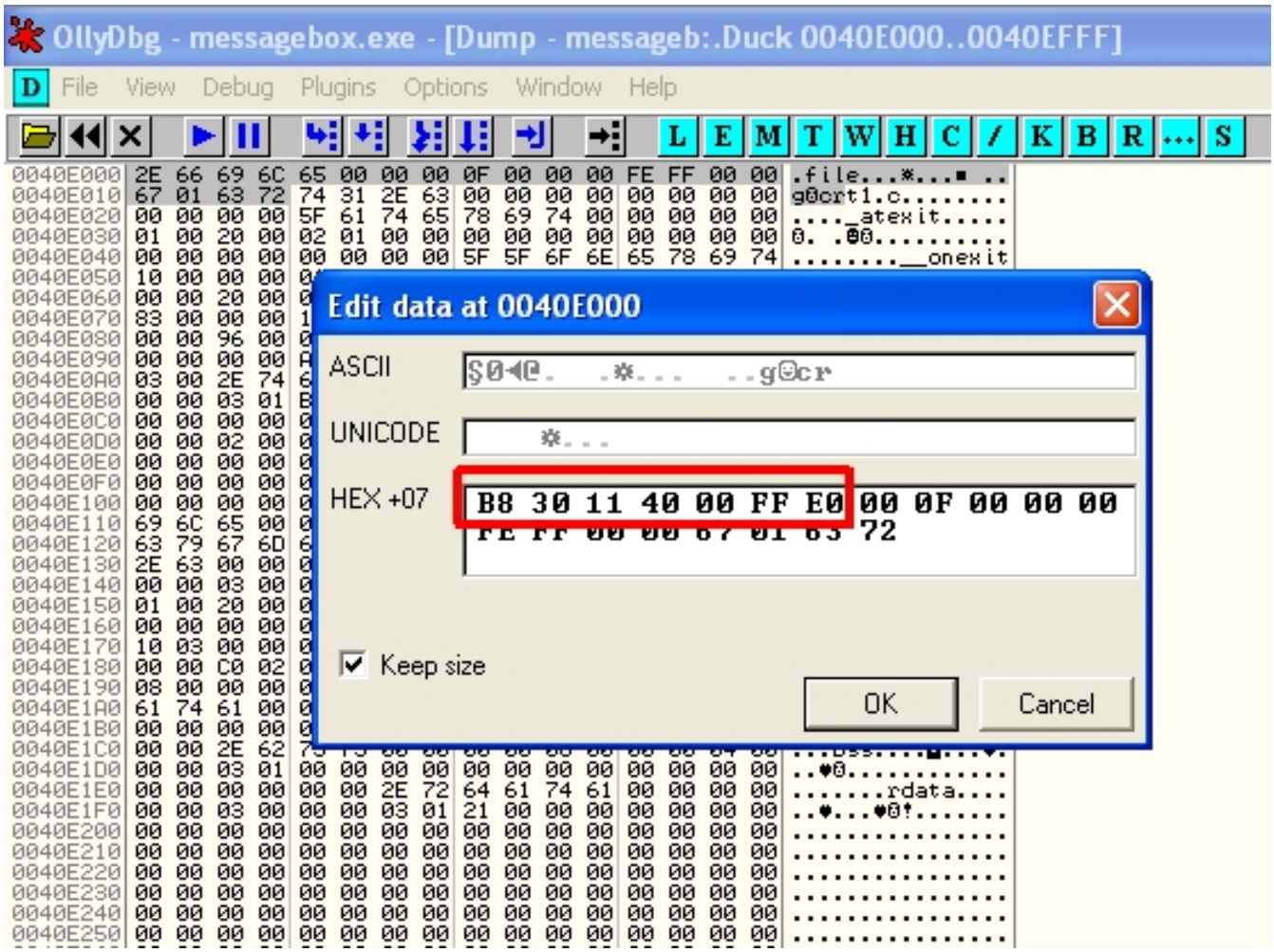


Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00001000				Priv	RW	RW	
00020000	00001000				Priv	RW	RW	
0022C000	00001000				Priv	RW Gua	RW	
0022D000	00003000			stack of ma	Priv	RW Gua	RW	
00230000	00003000				Map	R	R	
00240000	00004000				Priv	RW	RW	
00340000	00006000				Priv	RW	RW	
00350000	00003000				Map	RW	RW	
00360000	00016000				Map	R	R	\Device\HarddiskVolume1\WINDOW
00380000	00041000				Map	R	R	\Device\HarddiskVolume1\WINDOW
003D0000	00006000				Map	R	R	\Device\HarddiskVolume1\WINDOW
003E0000	00004000				Priv	RW	RW	
003F0000	00003000				Map	R	R	\Device\HarddiskVolume1\WINDOW
00400000	00001000	messageb		PE header	Imag	R	RWE	
00401000	00001000	messageb	.text	code	Imag	R	RWE	
00402000	00001000	messageb	.data	data	Imag	R	RWE	
00403000	00001000	messageb	.rdata		Imag	R	RWE	
00404000	00001000	messageb	.bss		Imag	R	RWE	
00405000	00001000	messageb	.idata	imports	Imag	R	RWE	
00406000	00001000	messageb	/4		Imag	R	RWE	
00407000	00001000	messageb	/19		Imag	R	RWE	
00408000	00001000	messageb	/35		Imag	R	RWE	
00409000	00001000	messageb	/47		Imag	R	RWE	
0040A000	00001000	messageb	/61		Imag	R	RWE	
0040B000	00001000	messageb	/73		Imag	R	RWE	
0040C000	00001000	messageb	/86		Imag	R	RWE	
0040D000	00001000	messageb	/97		Imag	R	RWE	
0040E000	00001000	messageb	.Duck		Imag	R	RWE	
00410000	00041000				Map	R	R	\Device\HarddiskVolume1\WINDOW
00460000	00041000				Map	R	R	
004B0000	00006000				Map	R E	R E	
00570000	00002000				Map	R E	R E	
00580000	00103000				Map	R	R	
00690000	00001000				Priv	RW	RW	
006A0000	000CB000				Map	R E	R E	
009A0000	00001000				Priv	RW	RW	
76370000	00001000	IMM32		PE header	Imag	R	RWE	
76371000	00015000	IMM32	.text	code,import	Imag	R	RWE	
76386000	00001000	IMM32	.data	data	Imag	R	RWE	
76387000	00005000	IMM32	.rsrc	resources	Imag	R	RWE	
7638C000	00001000	IMM32	.reloc	relocations	Imag	R	RWE	
77C00000	00001000	msvcrt		PE header	Imag	R	RWE	
77C01000	0004C000	msvcrt	.text	code,import	Imag	R	RWE	
77C4D000	00007000	msvcrt	.data	data	Imag	R	RWE	

[9]
Nejprve stiskneme ikonu s písmenem M, potom dvojklikneme na poslední sekci a jsme tam :)

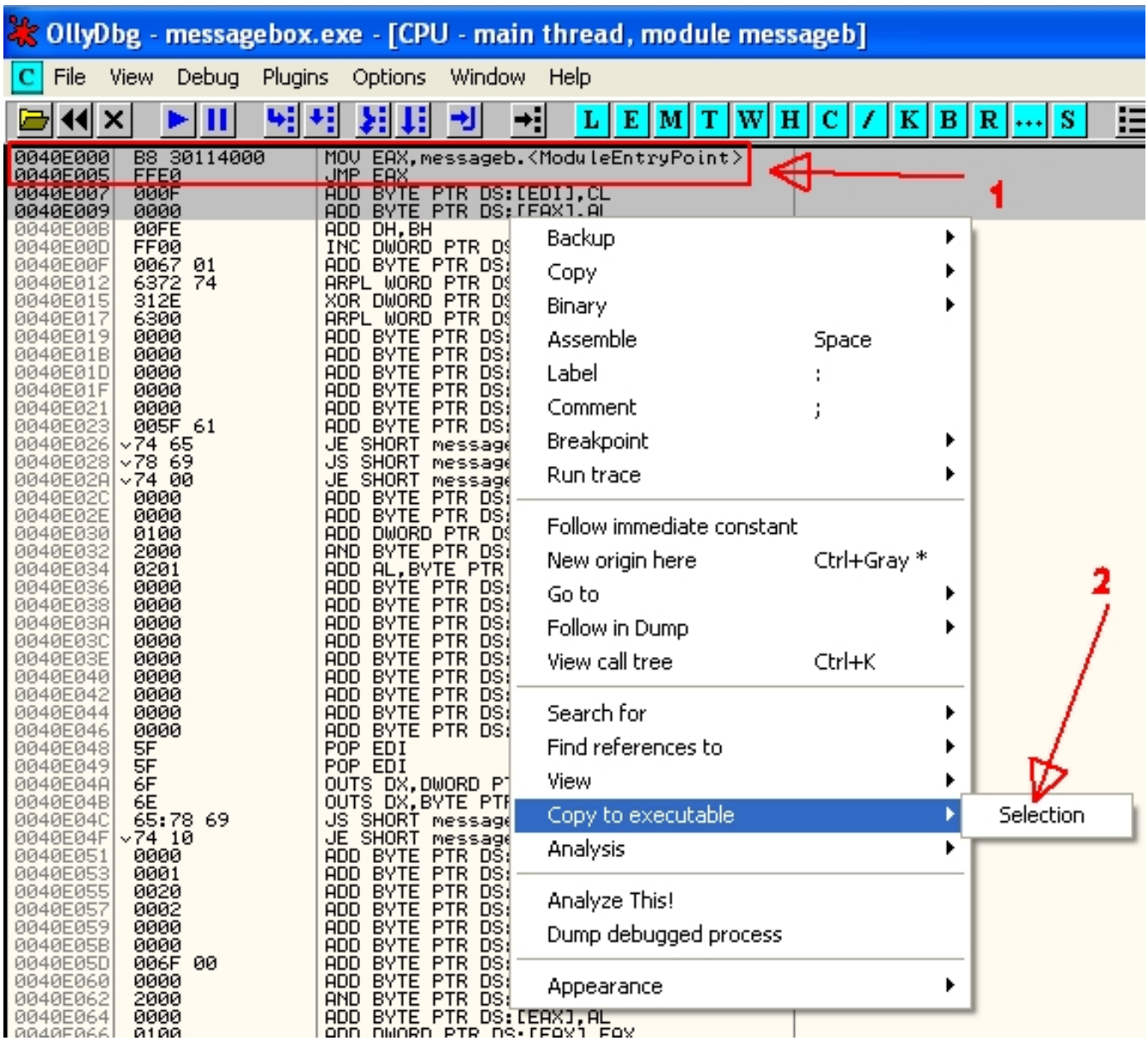
Vlastnostmi a možnostmi OllyDbg se v tomto seriálu věnovat nebudu. Pro naše účely bohatě bude

stačit otrocké použití. Pokud někdo ale bude mít zájem, určitě doporučuji osvojit si ovládání OllyDbg, protože se jedná o výborný debugger.
Otevřeme OllyDbg a přetáhneme do něj naši aplikaci. V liště stiskneme tlačítko M na zeleném podkladu a přejdeme do okna Memory Map. Ve sloupci Section si najdeme sekci s názvem, který jste použili. V mém případě je to .Duck. Dvakrát na tuto sekci poklikáme. Vybereme okno s názvem Dump-..., označíme myší prvních sedm bajtů a stiskneme mezerník. Tímto jsme se dostali do editačního okna a přepíšeme do něj bajty popsané výše.



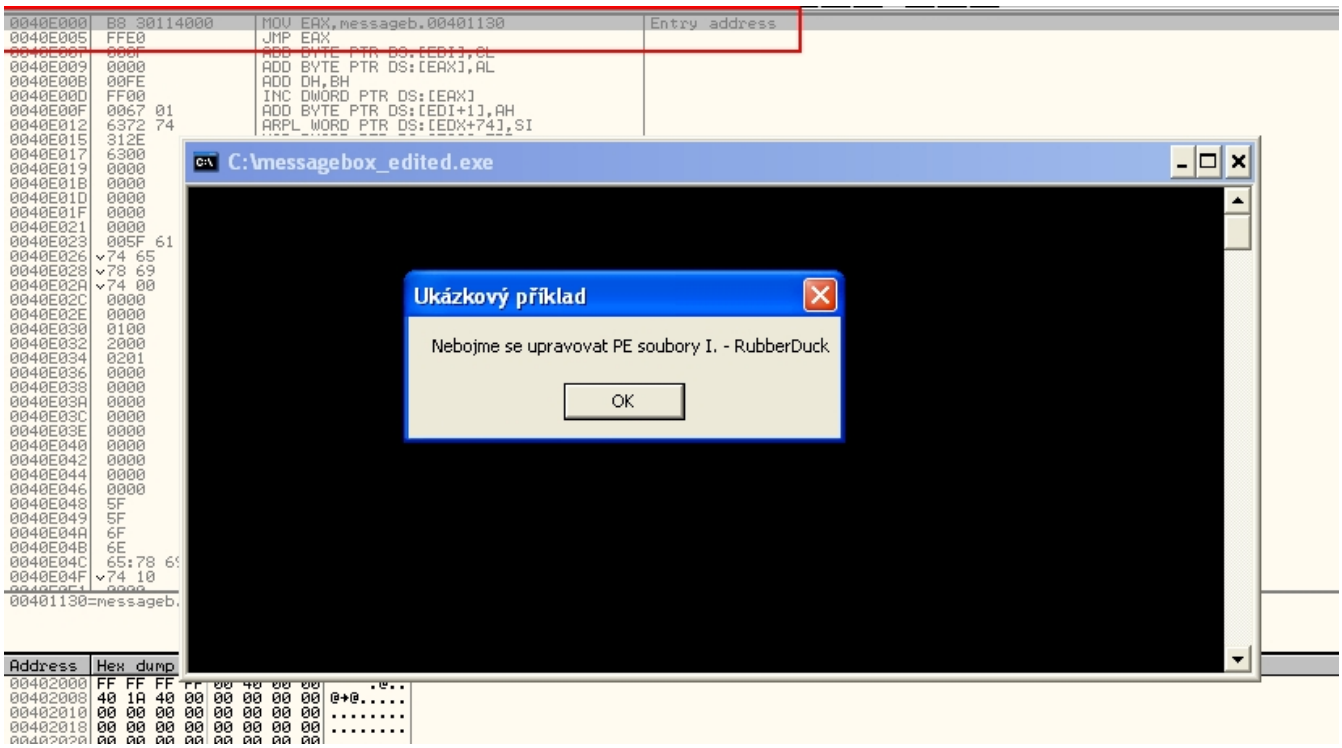
[10]
Označené bajty, které byly přepsány naším kódem

Po skončení editace stiskneme OK. Opět bychom se měli objevit v okně Dump. Všimněte si, že jsme na adrese 0040E000h. Na horní liště stiskneme C na zeleném podkladu a přistaneme v okně CPU. Zde stiskneme kombinaci kláves CTRL+G a vložíme hodnotu 0040E000. Po stisknutí OK se objevíme právě na adrese 40E000h. Zde si můžeme zkontrolovat kód assembleru, který jsme vložili. Pokud jsme neudělali chybu, měl by být vidět výše zmíněný kód assembleru. Označíme naše dva řádky, klikneme pravým tlačítkem myši do stejného okna a zvolíme možnost Copy to executable -> Selection.



[11]
Nejdříve označíme první dva řádky, které jsme upravili, a pak uložíme přes volbu Copy to executable

Objevíme se v okně File s označenými našimi dvěma řádky. Opět klikneme pravým tlačítkem myši a zvolíme možnost Save file. Pro jistotu přejmenujeme výsledný EXE soubor třeba na pokus.exe a uložíme. Nyní máme náš kód v PE souboru a můžeme testovat. Nejdříve zběžné otestování pomocí klasického spuštění. Et'voila! PE soubor funguje jako dřív :) Nyní si aplikaci otevřeme v OllyDbg. Oproti předchozímu spuštění na nás vybafne varovné okno. Olly zjistil, že Entry point neukazuje do sekce kódu a myslí si, že kód může být zapakovaný nebo jinak chráněný. Dáme OK a kód se načte. Pokud se podíváme na adresu, jedná se o adresu naší sekce. První dva řádky jsou nám rovněž povědomé :) Pokud nyní stiskneme klávesu F9, aplikace se spustí a bude normálně fungovat. Gratuluji! Právě jste upravili svou první PE binárku a vylepšili ji svým vlastním kódem :)



[12]
Upravená aplikace messagebox.exe běží :) V rámečku Entry Point aplikace s naším kódem

Závěr
V dnešním prvním dílu seriálu o změnách PE souborů jsme si ukázali, jak velmi jednoduše vložit do stávající aplikace novou sekci s vlastním kódem tak, aniž by to jakkoliv omezilo vykonávání. Rovněž jsme si ukázali, že použitá matematika je velice jednoduchá a nezadá si s tou ze třetí čtvrté třídy, tudíž by tuto operaci zvládl i žák prvního stupně základní školy. V příštím dílu si ukážeme, jak celý tento proces automatizovat a řekneme si o záludnostech, které jsem dnes schválně vynechal, jelikož nebylo třeba se jimi zabývat a plést hlavu. Každému, komu se tato činnost zalíbila, doporučuji začít studovat assembler. Zatím není třeba znát žádné speciální věci. Člověk si vystačí s cca pěti až deseti instrukcemi. Dále určitě přijde vhod debugger (Olly, Immunity nebo IdaPro, případně WinDbg). A hlavně: Zkoušet, přemýšlet, zkoušet, přemýšlet a zkoušet, protože pouze cvičení dělá mistry.

Odkazy
[1] - Matt Pietrek: An In-Depth Look into the Win32 Portable Executable File Format [13]
[2] - Matt Pieterek: Peering Inside the PE: A Tour of the Win32 Portable Executable File Format [14]
[3] - struktura IMAGE_DOS_HEADER [15]
[4] - struktura IMAGE_NT_HEADERS [16]
[5] - struktura IMAGE_FILE_HEADER [17]
[6] - struktura IMAGE_OPTIONAL_HEADER [18]
[7] - struktura IMAGE_SECTION_HEADER [19]
[8] - testovací messagebox.exe [20]
[9] - PE Tools [21]
[10] - OllyDbg [22]

Poznámka: Pokud by se někdo divil, proč jsem zvolil takový nepěkný příklad vloženého kódu: Nechtěl jsem čtenáře nechat přepisovat desítky hexadecimálních čísel jen proto, aby viděli MessageBox, když ho uvidí v dalším dílu :)

URL článku: https://security-portal.cz/clanky/nebojme-se-upravovat-pe-soubory-i

Odkazy:
[1] https://security-portal.cz/users/rubberduck
[2] https://security-portal.cz/category/tagy/cracking
[3] https://security-portal.cz/category/tagy/programming

[4] https://security-portal.cz/category/tagy/virus-worms
[5] http://security-portal.cz/sites/default/files/open_file.jpg
[6] http://bflow.security-portal.cz/images/sekce.jpg
[7] http://security-portal.cz/sites/default/files/new_sec.jpg
[8] http://security-portal.cz/sites/default/files/edit_sec.jpg
[9] http://security-portal.cz/sites/default/files/go_to_new_section.jpg
[10] http://security-portal.cz/sites/default/files/edit_section_code.jpg
[11] http://security-portal.cz/sites/default/files/save_edited_section.jpg
[12] http://security-portal.cz/sites/default/files/run.jpg
[13] http://msdn.microsoft.com/en-us/magazine/cc301805.aspx
[14] http://msdn.microsoft.com/en-us/library/ms809762.aspx
[15] http://www.nirsoft.net/kernel_struct/vista/IMAGE_DOS_HEADER.html
[16] http://msdn.microsoft.com/en-us/library/windows/desktop/ms680336%28v=vs.85%29.aspx
[17] http://msdn.microsoft.com/en-us/library/windows/desktop/ms680313%28v=vs.85%29.aspx
[18] http://msdn.microsoft.com/en-us/library/windows/desktop/ms680339%28v=vs.85%29.aspx
[19] http://msdn.microsoft.com/en-us/library/windows/desktop/ms680341%28v=vs.85%29.aspx
[20] http://bflow.security-portal.cz/down/messagebox.exe
[21] http://uinc.ru/files/neox/PE_Tools.shtml
[22] http://www.ollydbg.de/