

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

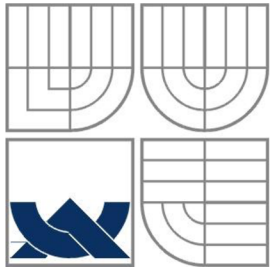
VISUAL BASIC DISASSEMBLER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

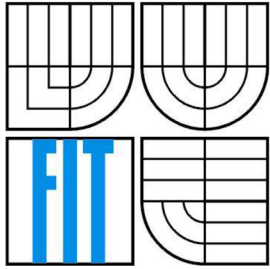
AUTOR PRÁCE
AUTHOR

BC. JÁN ADAMICA

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VISUAL BASIC DISASSEMBLER

VISUAL BASIC DISASSEMBLER

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. JÁN ADAMICA

VEDOUČÍ PRÁCE

SUPERVISOR

Ing. RADOVAN JOŠTH

BRNO 2011

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií

Akademický rok 2010/2011

Zadání diplomové práce

Řešitel: **Adamica Ján, Bc.**

Obor: Počítačové systémy a sítě

Téma: **Visual Basic disassembler**

Visual Basic Disassembler

Kategorie: Překladače

Pokyny:

1. Analyzujte binární formát Visual Basic-u.
2. Prostudujte a zorientujte se v p-code a native-code.
3. Navrhněte způsob analýzy a disassembling-u souborů Visual Basic-u.
4. Implementujte navržený disassembler.
5. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.

Literatura:

- dle pokynů vedoucího

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 - 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese

<http://www.fit.vutbr.cz/info/szz/>

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Jošth Radovan, Ing.**, UPGM FIT VUT

Datum zadání: 20. září 2010

Datum odevzdání: 25. května 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.



doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cílem této diplomové práce je navrhnout a implementovat disassembler binárních .exe souborů programovacího jazyka Visual Basic 6. Je zde popsán formát spustitelných přenositelných souborů, popsána funkce disassembleru, jakož i funkce Windows API volání. Visual Basic disassembler by měl uživateli poskytnout co nejvíce dostupných informací o souboru Visual Basic 6. V této práci je podrobně popsána struktura binárních souborů Visual Basic 6.

Abstract

Main goal of this thesis is to create disassembler for Visual Basic 6 binary .exe files. There is description of Portable Executable files, description of disassembler, as well as functions of Windows API calls. Visual Basic disassembler should provide as much information about Visual Basic 6 file as possible. There is detailed structure specification of Visual Basic 6 binary files.

Klíčové slová

Disassembler, Visual Basic, VB6, PE, Windows API, native kód, p-kód

Keywords

Disassembler, Visual Basic, VB6, PE, Windows API, native code, p-code

Citácia

Adamica Ján: Visual Basic disassembler, diplomová práce, Brno, FIT VUT v Brne, 2011

Visual Basic disassembler

Prehlásenie

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Ing. Radovana Joštha.

Ďalšie informácie mi poskytol pán Pavel Krčma.

Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Ján Adamica
25.mája 2011

© Ján Adamica, 2011.

Táto práca vznikla ako školské dielo na Vysokom učení technickom v Brne, Fakulte informačných technológií. Práca je chránená autorským zákonom a jej použitie bez udelenia oprávnenia autorom je nezákonné, s výnimkou zákonom definovaných prípadov.

Obsah

Obsah	1
1 Úvod.....	3
2 Formát prenositeľných spustiteľných súborov.....	4
2.1 MS-DOS Hlavička.....	4
2.2 MS-DOS Stub.....	6
2.3 PE hlavička.....	6
2.4 Tabuľka a hlavičky sekcií.....	9
2.5 Sekcie.....	10
2.6 Problémy PE formátu	11
3 Disassembler	13
4 Rozhranie pre programovanie aplikácií	15
4.1 Windows API.....	15
4.2 Windows API, ktoré využíva väčšina malware	15
4.3 Problémy spojené s Windows API	17
5 Binárny formát Visual Basic 6.....	18
5.1 Native kód.....	18
5.2 P-kód.....	18
5.3 Problémy binárneho formát Visual Basic 6.....	19
6 Použité programy	20
6.1 IDA Pro (Interactive disassembler Pro).....	20
6.2 OllyDbg	21
6.3 McAfee FileInsight.....	23
7 Návrh disassembleru	24
8 Analýza a implementácia.....	26
8.1 Získanie adresy Entry Point a iných základných informácií o PE súbore	28
8.2 Získanie Importov.....	32
8.3 Informácie o súbore Visual Basic 6.....	36
8.3.1 Visual Basic hlavička.....	36
8.3.2 Informácie o projekte.....	40
8.3.3 Sekundárne informácie o projekte	41
8.3.4 Tabuľka objektov	41
8.3.5 Verejný deskriptor objektu	43
8.3.6 Informácie o objekte	44
8.3.7 Privátny deskriptor objektu.....	45

8.3.8	COM Registračné dáta.....	46
8.3.9	COM Registračné informácie	47
8.3.10	COM Designer informácie.....	48
8.3.11	Prehľad štruktúry binárneho súboru Visual Basic 6	49
8.4	Praktické riešenie	51
8.5	Praktické využitie	54
9	Záver	55

1 Úvod

Cieľom tejto diplomovej práce je navrhnuť a implementovať disassembler binárnych .exe súborov programovacieho jazyka Visual Basic 6. Je tu opísaný formát spustiteľných prenositeľných súborov, popísaná funkcia disassemblera, ako aj funkcia Windows API volaní. Visual Basic disassembler by mal užívateľovi poskytnúť, čo najviac dostupných informácií o súbore Visual Basic 6.

V tejto diplomovej práci je predstavený formát prenosných spustiteľných (*Portable Executable*) súborov. Je tu opísaná štruktúra týchto súborov, ich hlavičky (*headers*), sekcie (*sections*), tabuľky adries (*import address table*) a rôzne iné informácie o prenosných spustiteľných súboroch.

Ďalej je tu predstavená funkcia disassembleru. Vysvetlenie najbežnejších inštrukcií používaných v jazyku symbolických inštrukcií (*assembly language*). Sú tu vysvetlené volania Windows API funkcií a použitie v aplikáciách pre Microsoft Windows. Je načrtnuté správanie najbežnejších typov škodlivých súborov (*malware*).

Je tu opísaný binárny formát programovacieho jazyka Visual Basic 6. Detailnejšie popísané verzie jeho výstupných binárnych súborov: native kód a p-kód.

Sú predstavené programy, ktoré slúžia na analýzu prenosných spustiteľných súborov: IDA (Interactive Disassembler), OllyDbg, NuMeGa SmartCheck a iné.

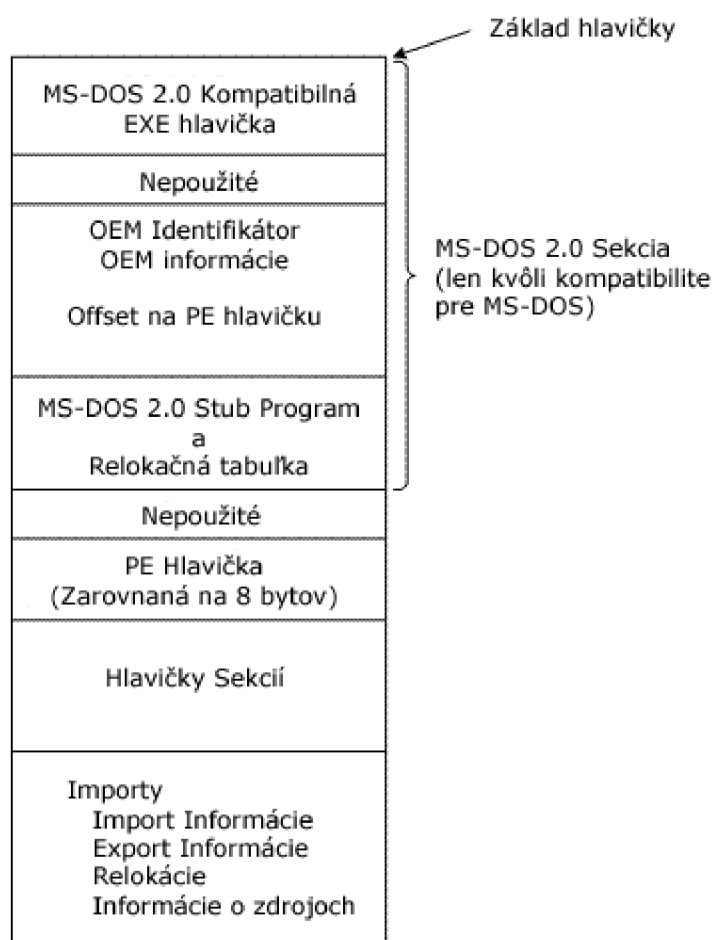
Návrh disassembleru ako aj spôsob jeho implementácie popisuje siedma kapitola.

V ďalšej kapitole je dopodrobna popísaný postup pri analýze štruktúr binárneho súboru Visual Basic 6. Je to popísaný výpočet vstupného bodu programu, získanie importov z DLL knižníc. Nasleduje podrobný popis a analýza štruktúr binárnych súborov Visual Basic 6. Táto kapitola je zakončená prehľadným grafom závislosti a prepojenie jednotlivých štruktúr binárnych súborov Visual Basic 6.

Ďalšia kapitola sa zaoberá popisom implementácie Disassembleru Visual Basic v programovacom jazyku C# vo vývojovom prostredí Microsoft Visual Studio 2010. Nasleduje popis hotového programu a vysvetlenie jednotlivých funkcií a prvkov programu.

2 Formát prenositeľných spustiteľných súborov

Príchodom OS Windows 3.1 Microsoft uviedol nový formát spustiteľných súborov, ktoré sa nazývajú prenositeľné spustiteľné súbory (*Portable Executable files*). Názov prenositeľné spustiteľné popisuje fakt, že formát týchto súborov nie je určený pre špecifickú architektúru. Na obrázku 2.1 je schéma typického prenositeľného spustiteľného súboru. [2]



Obr. 2.1 Schéma typického prenositeľného EXE súboru [1]

2.1 MS-DOS Hlavička

Ako vidíme na obrázku 2.1 prvá časť prenositeľného spustiteľného (PE) súboru je tvorená MS-DOS hlavičkou. MS-DOS hlavička nie je v PE súbore žiadna novinka. Táto hlavička je rovnaká od čias

operačného systému MS-DOS 2.0. Dôvod prečo sa táto hlavička stále vyskytuje v PE súboroch je ten, že keď daný súbor spustíme v operačnom systéme Windows 3.1 alebo predchádzajúcich verziách až po MS-DOS 2.0, operačný systém môže súbor prečítať a zhodnotiť či je kompatibilný. Keď sa pokúsime spustiť súbor určený pre Windows NT pod operačným systémom MS-DOS, dostaneme správu o nekompatibilite. Ak by MS-DOS hlavička nebola na prvom mieste v PE súbore operačný systém by tento súbor jednoducho nebol schopný spustiť. MS-DOS hlavička zaberá prvých 64 bytov PE súboru. Jej štruktúra je znázornená tu:

```
typedef struct _IMAGE_DOS_HEADER { // DOS .EXE hlavička
    USHORT e_magic;           // Magické číslo
    USHORT e_cblp;           // Byty na poslednej stránke súboru
    USHORT e_cp;             // Stránok v súbore
    USHORT e_crlc;          // Relokácie
    USHORT e_cparhdr;       // Veľkosť hlavičky v odsekoch
    USHORT e_minalloc;      // Minimum potrebných extra odsekov
    USHORT e_maxalloc;      // Maximum potrebných extra odsekov
    USHORT e_ss;            // Počiatočná (relatívna) SS hodnota
    USHORT e_sp;            // Počiatočná SP hodnota
    USHORT e_csum;          // Kontrolná suma
    USHORT e_ip;            // Počiatočná IP hodnota
    USHORT e_cs;            // Počiatočná (relatívna) CS hodnota
    USHORT e_lfarlc;        // Adresa relokačnej tabuľky
    USHORT e_ovno;          // Číslo prekrývania
    USHORT e_res[4];        // Rezervované slová
    USHORT e_oemid;         // OEM identifikátor (pre e_oeminfo)
    USHORT e_oeminfo;       // OEM informácie; e_oemid špecifické
    USHORT e_res2[10];      // Rezervované slová
    LONG e_lfanew;          // Adresa novej EXE hlavičky
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER; [2]
```

00000000	4D 5A 90 00 03 00 00 00	04 00 00 00 FF FF 00 00	MZ.....
00000010	B8 00 00 00 00 00 00 00	40 00 00 00 00 00 00 00@.....
00000020	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000003C	00 00 00 00 00 00 00 00	00 00 00 00 E8 00 00 00[.....
0000004E	0E 1F BA 0E 00 B4 09 CD	21 B8 01 4C CD 21 54 68!..L.!Th
00000050	69 73 20 70 72 6F 67 72	61 6D 20 63 61 6E 6E 6F	is program canno
00000060	74 20 62 65 20 72 75 6E	20 69 6E 20 44 4F 53 20	t be run in DOS
00000074	6D 6F 64 65 2E 0D 0D 0A	24 00 00 00 00 00 00 00	mode.....\$.....
00000080	83 C2 32 29 C7 A3 5C 7A	C7 A3 5C 7A C7 A3 5C 7A	..2).. \z.. \z.. \z
00000090	CE DB D8 7A C6 A3 5C 7A	CE DB C9 7A C5 A3 5C 7A	..z.. \z...z.. \z
000000A0	CE DB CF 7A DA A3 5C 7A	C7 A3 5D 7A 33 A3 5C 7A	..z.. \z..]z3. \z
000000B0	CE DB DF 7A D3 A3 5C 7A	CE DB D5 7A CC A3 5C 7A	..z.. \z...z.. \z
000000C0	CE DB C8 7A C6 A3 5C 7A	CE DB CD 7A C6 A3 5C 7A	..z.. \z...z.. \z
000000D0	52 69 63 68 C7 A3 5C 7A	00 00 00 00 00 00 00 00	Rich.. \z.....
000000E8	00 00 00 00 00 00 00 00	50 45 00 00 64 86 06 00PE...d...

Obr. 2.2 HEX kód PE súboru notepad.exe

Na obrázku 2.2 vidíme hneď na začiatku na adrese 0x00 magické číslo DOS .EXE hlavičky. Na adrese 0x3C vidíme adresu novej hlavičky PE súboru, ktorá sa nachádza na adrese 0xE8. Toto sú najdôležitejšie údaje z MS-DOS hlavičky a len tieto sú v nej povinné. [3]

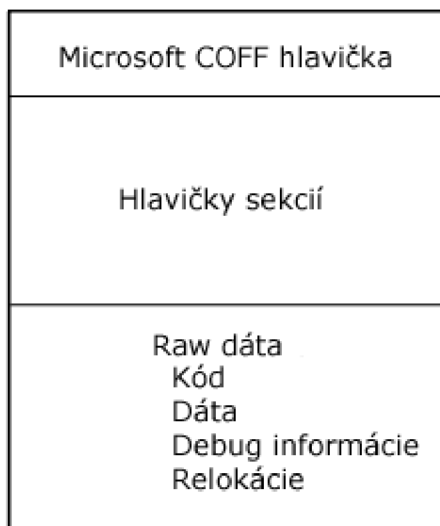
2.2 MS-DOS Stub

Je aplikácia, ktorá je schopná bežať pod operačným systémom MS-DOS. Je umiestnená na začiatku EXE súboru. Prekladač tu umiestňuje predvolený stub, ktorý, ak je spustený pod operačným systémom MS-DOS, vypíše na obrazovku správu: „This program cannot be run in DOS mode“ vid' obrázok 2.2. Túto správu môže programátor v niektorých prekladačoch zmeniť na ľubovoľnú. Na adrese 0x3C má stub zapísaný ofset na PE signatúru. Táto informácia umožňuje systému Microsoft Windows spustiť tento súbor aj napriek MS-DOS stub. [1]

2.3 PE hlavička

PE hlavička pozostáva z PE signatúry, COFF (*Common Object File Format*) objektu, ktorý pozostáva z hlavičky a nepovinných hlavičiek. Za týmito hlavičkami nasledujú hlavičky sekcií.

PE signatúra je 4 bajtová signatúra pozostávajúca z písmen „P“ a „E“ nasledovaných dvoma nulovými bajtmi. [1]



Obr. 2.3 Schéma typického COFF objektu

Zoznam signatúr PE súborov:

```
#define IMAGE_DOS_SIGNATURE          0x5A4D    // MZ
#define IMAGE_OS2_SIGNATURE          0x454E    // NE
#define IMAGE_OS2_SIGNATURE_LE      0x454C    // LE
```

```
#define IMAGE_NT_SIGNATURE 0x00004550 // PE00 [2]
```

Hneď po signatúre nasleduje štandardná COFF hlavička vid' tabuľka 2.1.

Ofset	Veľkosť	Pole	Popis
0	2	Machine	Číslo identifikujúce cieľovú architektúru 0x14c pre Intel 386 kompatibilné procesory 0x8664 pre 64 bitové procesory
2	2	NumberOfSections	Počet sekcií. Indikuje aj veľkosť tabuľky sekcií, ktorá nasleduje ihneď po hlavičkách.
4	4	TimeDateStamp	Spodných 32 bitov počtu sekúnd od 00:00 1. januára, 1970 (v C prostredí je to time_t konštanta), toto nám indikuje kedy bol súbor vytvorený.
8	4	PointerToSymbolTable	Ofset na COFF tabuľku symbolov, alebo nula ak tabuľka neexistuje. Zvyčajne je táto hodnota nula.
12	4	NumberOfSymbols	Počet záznamov v COFF tabuľke symbolov. Zvyčajne nula.
16	2	SizeOfOptionalHeader	Veľkosť nepovinnnej hlavičky, ktorá je potrebná pre spustiteľné súbory.
18	2	Characteristics	Príznamy, ktoré indikujú vlastnosti súboru.

Tabuľka 2.1 Formát štandardnej COFF hlavičky [1]

Dalších 224 bajtov (240 bajtov pri 64 bitovom programe) tvorí nepovinná hlavička, ktorá aj napriek svojmu názvu je pri spustiteľných súboroch povinná. Táto nepovinná hlavička obsahuje najdôležitejšie informácie o spustiteľnom súbore ako napríklad počiatková veľkosť zásobníka, vstupný bod programu (*entry point*), preferovanú základnú adresu (*base address*), verziu operačného systému, informácie o zarovnaní sekcií, verzia kompilátora, veľkosť spustiteľného kódu a iné. [1][2]

Štruktúra nepovinnnej hlavičky je znázomená tu:

```
typedef struct _IMAGE_OPTIONAL_HEADER {
    //
    // Standard fields.
    //
    USHORT   Magic;
    UCHAR    MajorLinkerVersion;
    UCHAR    MinorLinkerVersion;
    ULONG    SizeOfCode;
    ULONG    SizeOfInitializedData;
    ULONG    SizeOfUninitializedData;
```



```

        ULONG    AddressOfEntryPoint;
        ULONG    BaseOfCode;
        ULONG    BaseOfData;
        //
        // NT additional fields.
        //
        ULONG    ImageBase;
        ULONG    SectionAlignment;
        ULONG    FileAlignment;
        USHORT   MajorOperatingSystemVersion;
        USHORT   MinorOperatingSystemVersion;
        USHORT   MajorImageVersion;
        USHORT   MinorImageVersion;
        USHORT   MajorSubsystemVersion;
        USHORT   MinorSubsystemVersion;
        ULONG    Reserved1;
        ULONG    SizeOfImage;
        ULONG    SizeOfHeaders;
        ULONG    CheckSum;
        USHORT   Subsystem;
        USHORT   DllCharacteristics;
        ULONG    SizeOfStackReserve;
        ULONG    SizeOfStackCommit;
        ULONG    SizeOfHeapReserve;
        ULONG    SizeOfHeapCommit;
        ULONG    LoaderFlags;
        ULONG    NumberOfRvaAndSizes;
        IMAGE_DATA_DIRECTORY
DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
    } IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER; [2]

```

DataDirectory je pole štruktúr IMAGE_DATA_DIRECTORY. Celkovo má 16 členov. DataDirectory obsahuje ofsety a veľkosti dôležitých dátových štruktúr PE súborov. Každý jeho člen obsahuje informácie o dôležitej dátovej štruktúre. [24] Nie všetky tieto dátové štruktúry sú v PE súbore povinné.

Štruktúra IMAGE_DATA_DIRECTORY je znázornená tu:

```

typedef struct _IMAGE_DATA_DIRECTORY {
    DWORD VirtualAddress;
    DWORD Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;

```

Člen	Ofset (PE/PE32+)	Popis
0	96/112	Tabuľka Exportov, adresa a veľkosť
1	104/120	Tabuľka Importov, adresa a veľkosť
2	112/128	Tabuľka Resource, adresa a veľkosť
3	120/136	Tabuľka výnimiek, adresa a veľkosť
4	128/144	Certifikačná tabuľka, adresa a veľkosť
5	136/152	Tabuľka Relokácií, adresa a veľkosť
6	144/160	Debug informácie, adresa a veľkosť
7	152/168	Dáta špecifikácie architektúry, adresa a veľkosť
8	160/176	Globálny register, ukazateľ, adresa a veľkosť
9	168/184	Tabuľka Thread local storage (TLS), adresa a veľkosť
10	176/192	Tabuľka počiatočnej konfigurácie, adresa a veľkosť
11	184/200	Tabuľka bound importov, adresa a veľkosť
12	192/208	Tabuľka import adries, adresa a veľkosť
13	200/216	Deskriptor oneskorených importov, adresa a veľkosť
14	208/224	CLR hlavička, adresa a veľkosť
15	216/232	Rezervované

Tabuľka 2.2 Dátové adresáre PE súboru [25]

2.4 Tabuľka a hlavičky sekcií

V sekciách sa nachádza vlastný obsah súboru, vrátane kódu, dát, prostriedkov (*resources*) a spustiteľných informácií. Každá sekcia má hlavičku a telo, ktoré obsahuje dáta. Sekcie môžu byť organizované akokoľvek ich kompilátor preloží, pokiaľ ich hlavička obsahuje dostatok informácií na ich spracovanie.

Každý riadok tabuľky sekcií je v skutočnosti hlavička sekcie. Táto tabuľka nasleduje ihneď po nepovinnnej hlavičke. Toto umiestnenie je dôležité, pretože hlavička súboru neobsahuje žiadne informácie o umiestnení tabuľky sekcií. Počet sekcií v tabuľke je určený poľom `NumberOfSections` v COFF hlavičke. Sekcie sú číslované od jednotky. Poradie kódovej a dátovo pamäťovej sekcie určuje kompilátor. Každá hlavička sekcií má nasledovný formát, 40 bajtov, vid' tabuľka 2.2. [1][2][23]

Ofset	Veľkosť	Pole	Popis
0	8	Name	8 bajtový UTF-8 reťazec dorovnaný nulami.
8	4	VirtualSize	Celková veľkosť sekcie v pamäti. Ak je hodnota väčšia ako hodnota SizeOfRawData, sekcia je dorovnaná nulami.
12	4	VirtualAddress	Adresa prvého bajtu sekcie vzhľadom k ImageBase v nepovinnnej tabuľke, keď je sekcia načítaná v pamäti.
16	4	SizeOfRawData	Veľkosť dát na disku. Musí byť násobkom FileAlignment z nepovinnnej hlavičky.
20	4	PointerToRawData	Ukazateľ na prvú stránku sekcií v COFF súbore. Musí byť násobkom FileAlignment z nepovinnnej hlavičky.
24	4	PointerToRelocations	Ukazateľ na začiatok relokácií pre sekciu. V PE súbore 0.
28	4	PointerToLinenumbers	Ukazateľ na začiatok line-number položku. V PE súbore 0.
32	2	NumberOfRelocations	Počet relokácií v sekcii. V PE súbore 0.
34	2	NumberOfLinenumbers	Počet riadkov v sekcii. V PE súbore 0.
36	4	Characteristics	Príznamy, ktoré indikujú vlastnosti sekcie.

Tabuľka 2.2 Formát hlavičky sekcie [1]

2.5 Sekcie

Aplikácia pre Windows NT má zvyčajne 9 preddefinovaných sekcií pomenovaných .text, .bss, .rdata, .data, .rsrc, .edata, .idata, .pdata, a .debug. Niektoré aplikácie nepotrebujú všetky tieto sekcie a niektoré si definujú aj ďalšie sekcie, ktoré potrebujú.

Spustiteľná sekcia s kódom, .text

Sekcia .text obsahuje vstupný bod programu. Tabuľka adres importov sa nachádza tiež v sekcii .text hneď za vstupným bodom (je to výhodné z toho dôvodu, že táto tabuľka obsahuje skokové inštrukcie). Keď sú Windows procesy načítané v adresovom priestore procesu, tabuľka adres importov sa upraví tak aby obsahovala fyzickú adresu každej importovanej funkcie.

Dátové sekcie, .bss, .rdata, .data

Sekcie obsahujúce rôzne dáta a premenné programu.

Sekcia so zdrojmi, .rsrc

.rsrc sekcia obsahuje informácie o zdrojoch daného súboru. Začína adresárom zdrojov ako aj ostatné sekcie, ale tie sú ďalej štruktúrované do stromu zdrojov.

Dátová sekcia s exportmi, .edata

Tabuľka adries exportov (*Export Address Table*)

Tabuľka adries exportov obsahuje adresy exportovaných vstupných bodov programu a exportovaných dát. Poradové číslo sa používa ako index tejto tabuľky. Každý záznam v tabuľke adries exportov je pole, ktoré používa dva formáty. Keď sa špecifikovaná adresa nenachádza v export sekcii (ktorá je definovaná adresou a dĺžkou v nepovinnej hlavičke), tak je to export RVA, čo je aktuálna adresa v kóde alebo dátach. Inak toto pole obsahuje RVA, ktoré označuje symbol alebo meno v inom DLL.

Tabuľka ukazateľov mien exportov (*Export Name Pointer Table*)

Tabuľka ukazateľov mien exportov je pole adries RVA ktoré ukazujú na tabuľku mien exportov.

Tabuľka poradia exportov (*Export Ordinal Table*)

Tabuľka poradia exportov je pole 16 bitových indexov na tabuľku adries exportov.

Tabuľka mien exportov (*Export Name Table*)

Tabuľka mien exportov obsahuje reťazce, ktoré ukazujú na tabuľku ukazateľov mien exportov. Reťazce v tejto tabuľke sú verejné a iné programy ich môžu použiť na importovanie symbolov. Každý exportovaný symbol má poradové číslo, ktoré je aj indexom tabuľky adries exportov. Mená exportov môžu mať len niektoré, žiadne alebo všetky exportované symboly. Exportované symboly, ktoré majú mená exportov používajú tabuľku poradia exportov a tabuľku ukazateľov mien exportov na priradenie poradia.

Dátová sekcia s importmi, .idata

Táto sekcia obsahuje importované dáta, zahŕňa aj adresár importov a tabuľku adries mien importov.
[1][2][21]

2.6 Problémy PE formátu

V kapitole 2. Sme si popísali formát PE súborov, ich hlavičiek a sekcií, dátových štruktúr. Väčšina bežných kompilátorov tieto štandardy dodržiava, avšak hlavne pri súboroch malware môže byť veľká časť údajov v hlavičkách, sekciách a dátových štruktúrach poškodená alebo zámerne upravená. Týmito poškodeniami a úpravami sa PE súbor javí ako nefunkčný a prakticky znemožňuje akúkoľvek

statickú analýzu týchto PE súborov, pretože mnohé nástroje na statickú analýzu nie sú na chybné informácie pripravené a vyhlásia súbor za chybný a nefunkčný. Tieto problémy sa veľmi často vyskytujú aj pri súboroch zabalených pomocou rôznych PE packerov a protektorov. Tieto nástroje veľmi často zámerne upravujú údaje v hlavičkách a sekciách, tak aby neobsahovali štandardné informácie a tým znemožnili analýzu týchto súborov pomocou statickej analýzy. [24]

Formát PE súborov má teda svoje špecifikácie, ale tieto sú veľmi často nedodržiavané a porušované. Napriek tomu však Windows Loader mnohé tieto chyby ignoruje a aj poškodený PE súbor spustí. Je teda na šikovnosti autora konkrétnych PE súborov, aké dobré má vedomosti o tom, ktoré položky hlavičiek a sekcií môžu obsahovať chybné informácie a PE súbor bude aj napriek tomu v OS Windows úspešne spustený.

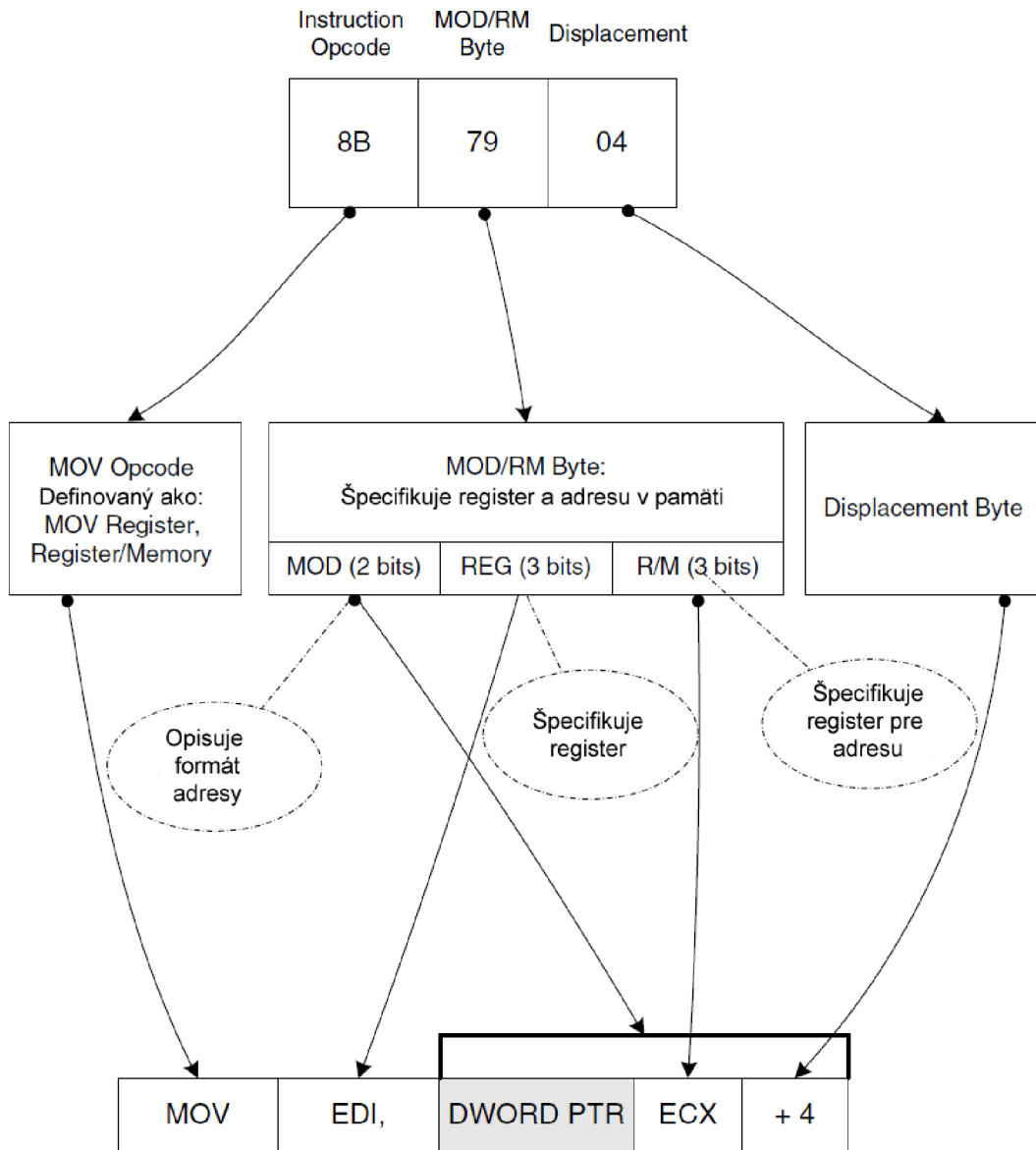
3 Disassembler

Sú prípady, keď potrebujeme zdrojový kód počítačového programu ale ten nie je k dispozícii. Napríklad potrebujeme upraviť nejaký starší firemný program od ktorého nemáme zdrojový kód. Alebo chceme aplikáciu prerobiť pre iný operačný systém. Alebo ak sme napadnutí útokom nejakého neznámeho počítačového vírusu alebo trojanu, tak chceme zistiť aké škody nám spôsobil. Výhodným nástrojom pre vyššie spomenuté úkony môže byť práve disassembler.

Disassembler je program, ktorého vstupom je binárny spustiteľný súbor programu a výstupom text obsahujúci kód jazyka symbolických inštrukcií celého programu alebo jeho častí. Toto je relatívne jednoduchý proces, ak berieme do úvahy, že jazyk symbolických inštrukcií je len binárny kód prevedený do textovej podoby. Disassembling je proces špecifický pre každý typ procesora, avšak existujú disassemblery, ktoré podporujú viacero CPU architektúr. Kvalitný disassembler je kľúčový komponent v nástrojoch reverzného inžiniera. [5][8]

Najznámejšie disassemblery sú IDA Pro (Interactive Disassembler Pro), OllyDbg, ILDASM a Texe. Prvé dva sú bližšie popísané v 6. kapitole, Použité programy.

Obrázok 3.1 ukazuje, ako disassembler prevádza postupnosť IA-32 postupností bajtov do človeku čitateľného jazyka symbolických inštrukcií. Tento proces typicky začína tak, že disassembler porovná postupnosti bajtov z tabuľkou, ktorá obsahuje zoznam všetkých inštrukcií a ich názvov (v tomto prípade kód 8B označuje inštrukciu MOV) spolu s ich formátmi. IA-32 inštrukcie sú ako funkcie, čo znamená, že každá inštrukcia má inú sadu "parametrov" (zvyčajne nazývaných operandy). Disassembler ďalej analyzuje, ktoré operandy sú použité v tejto osobitnej inštrukcii. [5][9]



Obr. 3.1 Preklad IA-32 inštrukcie z binárneho kódu do človeku zrozumiteľného jazyka symbolických inštrukcií [5]

4 Rozhranie pre programovanie aplikácií

Rozhranie pre programovanie aplikácií, alebo často používaná skratka API. Tento termín je používaný v programovaní. Ide o zbierku funkcií a tried (ale aj iných programov), ktoré určujú akým spôsobom sa majú funkcie knižníc volať zo zdrojového kódu programu. API funkcie sú programové celky, ktoré programátor volá namiesto vlastného naprogramovania.[11] Slúži ako rozhranie medzi rozličnými softvérovými programami a uľahčuje ich interakcie, podobne ako užívateľské rozhranie zjednodušuje interakciu medzi ľuďmi a počítačmi.

4.1 Windows API

Rozhranie pre programovanie aplikácií (API) operačného systému Windows umožňuje aplikáciám naplno využívať potenciál operačného systému Windows. Pomocou tohto API je možné vyvíjať aplikácie, ktoré bežia bez problémov na všetkých verziách Windows, zatiaľ čo využívajú výhody a možnosti, ktoré sú unikátne pre každú verziu operačného systému Microsoft Windows. (Toto programovacie rozhranie bolo pôvodne nazývané Win32 API, ale názvom Windows API presnejšie odráža jeho korene v 16-bitových verziách Windows a jeho podporu na 64-bitových verziách Windows.) [6][10]

Rozdiely v implementácii programovacích prvkov závisia na schopnostiach operačného systému. Tieto rozdiely sú uvedené v dokumentácii týchto programovacích prvkov.

Windows API sa skladá z nasledujúcich funkčných kategórií:

- Správa a riadenie
- Diagnostika
- Grafika a multimédia
- Sieť
- Zabezpečenie
- Systémové služby
- Používateľské rozhranie systému Windows [11]

4.2 Windows API, ktoré využíva väčšina malware

Väčšina škodlivých súborov robí po spustení stále tie isté úkony. Samozrejme, že tieto úkony sa dajú vykonávať v rôznom poradí a rôznymi postupmi, ale v podstate ide stále o tie isté veci (skrývanie sa

pred užívateľom, spustenie sa po štarte operačného systému), ktoré môžeme kategorizovať podľa správania: [13]

1. Zápis do registrov systému Windows do položiek, ktoré sú spustené po štarte systému

Zvyčajne použité API funkcie:

- RegSetValue
- RegSetValueEx
- RegSetKeyValue

Položky registrov do ktorých sa zapisuje:

- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
- HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Userinit
- HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Shell
- HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Run
- HKCU\Software\Microsoft\Windows\CurrentVersion\Run
- HKLM\System\CurrentControlSet\Services
- HKLM\System\CurrentControlSet\Control\Session Manager\BootExecute
- HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options
- HKLM\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run
- HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run

Použitie: obecné

2. Kopírovanie súborov na miesta, kde ich bežný užívateľ nehľadá (Predpokladáme, že systém je nainštalovaný do C:\WINDOWS)

API funkcie:

- CopyFile
- CopyFileEx
- kombinácia ReadFile/ReadFileEx a WriteFile/WriteFileEx

Miesta na disku:

- C:\
- C:\WINDOWS*
- C:\WINDOWS\SYSTEM32*
- C:\RECYCLER
- C:\System Volume Information
- C:\Documents and settings\USERNAME\Local Settings\Temp
- C:\Documents and settings\Administrator\Local Settings\Temp
- C:\Documents and settings\USERNAME\Application Data
- C:\Documents and settings\Administrator\Local Settings\Application Data
- C:\Program Files\Microsoft
- C:\Program Files\Windows NT

A mnoho ďalších iných...

Použitie: worm, dropper

3. Kopírovanie súborov na miesta, odkiaľ ich systém Windows spúšťa automaticky

API funkcie:

- CopyFile
- CopyFileEx
- kombinácia ReadFile/ReadFileEx a WriteFile/WriteFileEx

Miesto na disku:

- C:\ Documents and settings\All Users\Microsoft\Windows\Start Menu\
- C:\ Documents and settings\%USER%\Microsoft\Windows\Start Menu\

Použitie: obecné

4. Zmena atribútov súborov na skryté, systémové

Api funkcie:

- SetFileAttributes
- SetFileAttributesTransacted
- SetFileInformationByHandle

Použitie: obecné, primitívne rootkit techniky

5. Pripájanie sa na rôzne serveri a následné sťahovanie ďalších škodlivých súborov

Použitie: downloader

6. Otvorenie lokálneho portu, na ktorom škodlivý súbor počúva a čaká na pripojenia

Použitie: backdoor

7. Krádež prihlasovacích údajov pomocou predstierania prihlásenia do aplikácie

Použitie: banker, password stealer, keylogger, obecné programy na monitorovanie užívateľov

4.3 Problémy spojené s Windows API

Takmer každý štandardný program na svoju funkcionálnosť využíva Windows API funkcie. Avšak ich použitie môže byť nie vždy tak jasné a viditeľné. Existuje mnoho techník ako používané API funkcie zamaskovať, tak aby nebolo zrejmé aké Windows API funkcie konkrétny program využíva. Tieto techniky sú veľmi rozšírené pri tvorbe malware súborov. Existujú dve najviac využívané techniky skrývania Windows API funkcií:

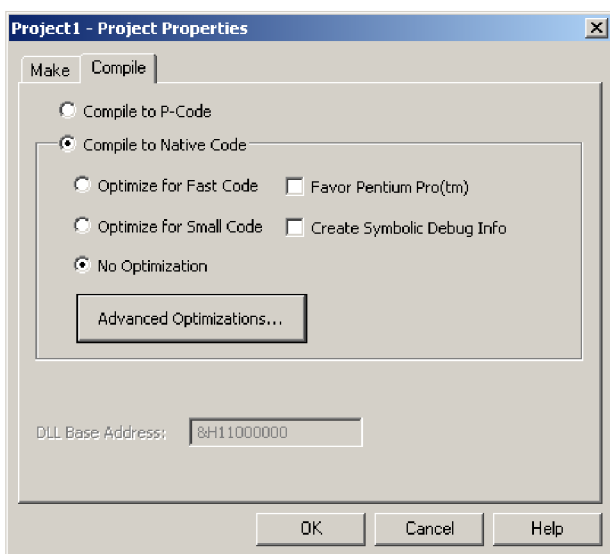
1. V tabuľke importov sa vyskytuje DLL knižnica, ktorej API funkcie sú používané avšak importovaná je len jedna funkcia, ktorá nie vždy musí byť podstatná. Windows Loader túto knižnicu načíta a pri behu programu potom načítavame ďalšie API funkcie konkrétnych DLL knižníc, ktoré potrebujeme.
2. Nenačítavame žiadne DLL knižnice a všetko hľadáme pomocou funkcie LoadLibrary pomocou konkrétnych adries a offsetov funkcií v pamäti Operačného systému.

Tieto metódy sú široko využívané aj rôznymi nástrojmi na zbalenie a obfuskovanie PE súborov. Vstupom týchto nástrojov je štandardne skompilovaný binárny súbor, ktorý je prebalený do „obálky“, ktorá sa stará o načítanie týchto API funkcií, využívaných v programe. [13]

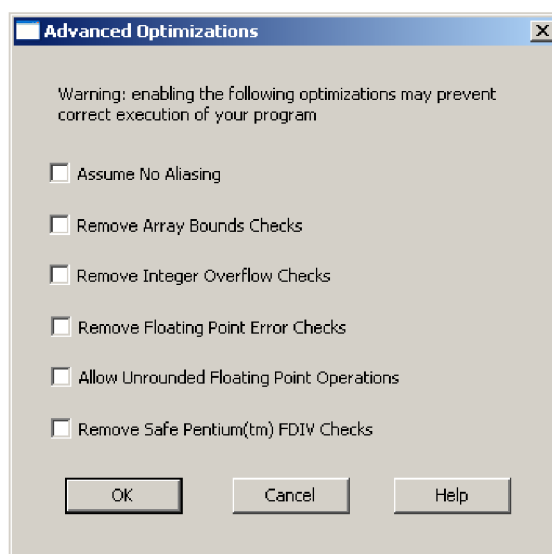
V tejto práci sa zaoberáme len štandardnými súbormi v rozbalenom referenčnom tvare.

5 Binárny formát Visual Basic 6

Pri kompilácii programov v prostredí Microsoft Visual Basic 6 máme na výber viaceru možností kompilácie. Jedná sa o možnosti kompilácie do p-kódu alebo native kódu bez optimalizácií alebo s optimalizáciami pre rýchlosť, poprípade veľkosť, obrázok 5.1. Pri kompilácii do native kódu môžeme zvoliť ešte ďalšie možnosti pre optimalizáciu, avšak tu Microsoft varuje, že pri aktivácii niektorých z nich už spustiteľný súbor nemusí fungovať, obrázok 5.2. [4]



Obr. 5.1 Možnosti kompilácie VB6



Obr. 5.2 Možnosti optimalizácie pre native kód

5.1 Native kód

Kompilácia projektu do native kódu znamená, že kód, ktorý píšeme bude plne skompilovaný do natívnych inštrukcií procesora, namiesto toho, aby bol skompilovaný do p-kódu. Tým sa výrazne urýchlia slučky a matematické výpočty, a môžu sa taktiež trochu urýchliť volania na služby poskytované MSVBVM60.DLL. Avšak, ani kompilácia do native kódu neodstraňuje potrebu tohto DLL súboru.

5.2 P-kód

P-kód, alebo pseudo kód, je medzistupeň medzi inštrukciami vysokej úrovni vo Visual Basic programe a nízkej úrovni natívneho kódu, ktorý spracováva procesor počítača. Priamo za behu Visual Basic prekladá každú inštrukciu p-kódu do natívneho kódu. Tieto p-kód inštrukcie sú menšie ako ekvivalentné inštrukcie natívneho kódu, čo veľmi výrazne znižuje veľkosť spustiteľného súboru. Ale pri spustení musí systém načítať do pamäte okrem programu aj interpretér p-kódu, a ten musí

dekódovať každú inštrukciu. Tým, že program skompilujeme priamo do native kódu, odstrániť tento medzi krok, kedy sa pri behu programu musí p-kód prekladať do native kódu. [14] [16]

5.3 Problémy binárneho formát Visual Basic 6

Nakoľko existujú rozličné metódy obfuskovania binárnych súborov Visual Basic 6, v tejto práci sa budeme zaoberať iba štandardnými súbormi, ktoré po výstupe z kompilátora nie sú nijako upravované. Týchto metód obfuskovania existuje nespočetné množstvo.

Najpoužívanejšou je metóda obfuskace na úrovni zdrojového kódu. Jedná sa o pregenerovanie zdrojového kódu do podoby, v ktorej nikdy nebol napísaný. Tento kód býva veľmi často doplnený rôznymi cyklami, funkciami, konštantami a reťazcami, ktoré v tomto kóde nemajú žiadnu funkcionality a slúžia len na zmetenie človeka, ktorý daný kód analyzuje.

Ďalšia metóda je, že na štandardne skompilovaný binárny súbor Visual Basic 6 sa pridá „obálka“, ktorá slúži na vytváranie autoreplikačných polymorfných .exe binárnych súborov. Pri tejto metóde sú textové reťazce nahradzované náhodne generovanými reťazcami. Tieto reťazce sa v binárnom súbore budú vyskytovať vždy na rovnakom mieste avšak budú sa líšiť.

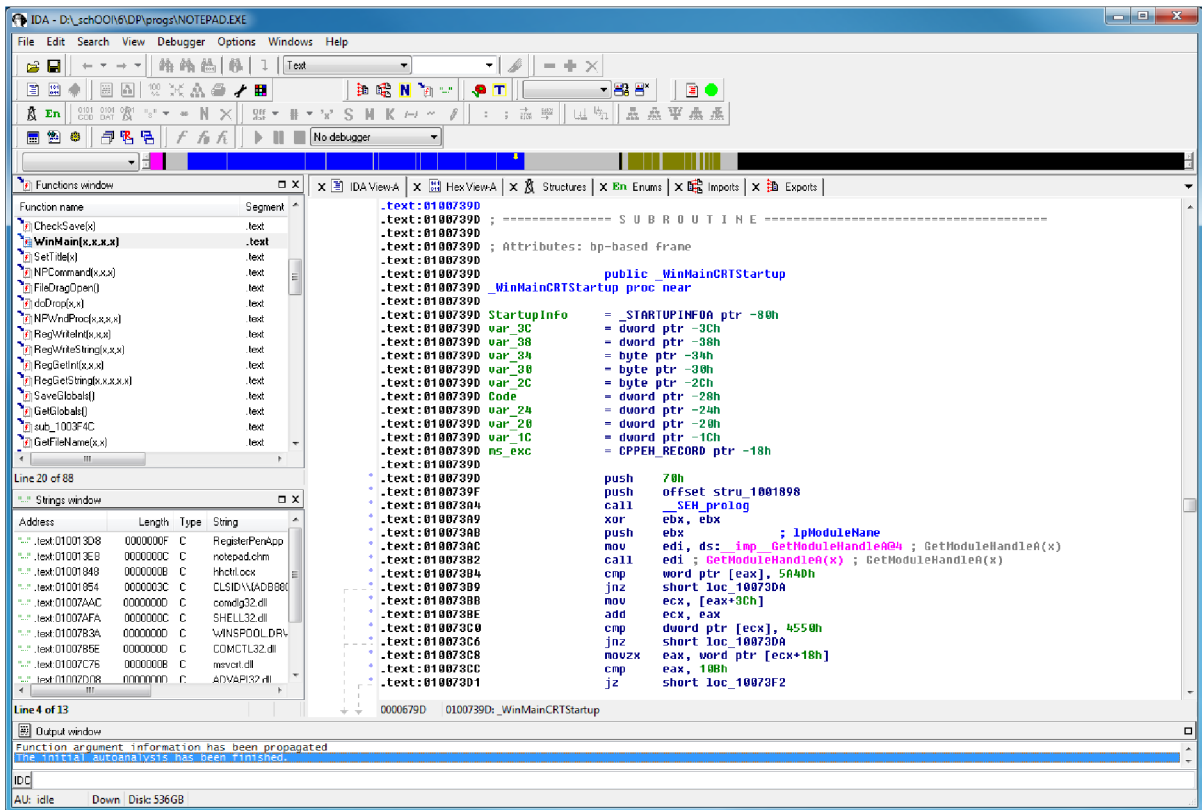
6 Použité programy

V nasledujúcom texte sú popísané niektoré dôležité programy použité pri analýze spustiteľných súborov skompilovaných v prostredí Visual Basic 6.

6.1 IDA Pro (Interactive disassembler Pro)

Interaktívny disassembler, viac známy ako IDA, je disassembler používaný pre reverzné inžinierstvo. Podporuje rôzne formáty spustiteľných súborov pre rôzne procesory a operačné systémy. Tiež môže byť použitý ako debugger pre prostredie Windows, Mac OS X Mach-O, a Linux ELF spustiteľné súbory.

IDA vykonáva oveľa automatickej analýzy kódu pomocou krížových odkazov medzi kódovými sekciami, znalosti parametrov API volaní a ďalších informácií. Avšak charakter disasemblingu je taký, že vyžaduje veľké množstvo ľudského zásahu. IDA má interaktívne funkcie, ktorými napomáha užívateľovi dokonale disassemblovať daný súbor. Typický IDA užívateľ začne s automaticky generovaným disassemblovaným kódom a potom prevádza rozličné sekcie kódu na dáta a naopak, pomenováva niektoré časti, komentuje, a inak dopĺňuje informácie do zoznamu, až pokiaľ mu nie je jasné, čo daný program robí. [7]



Obr. 6.1 Typická obrazovka IDA Pro, ukazujúce disasemblovaný kód, zoznam funkcií a zoznam reťazcov

6.2 OllyDbg

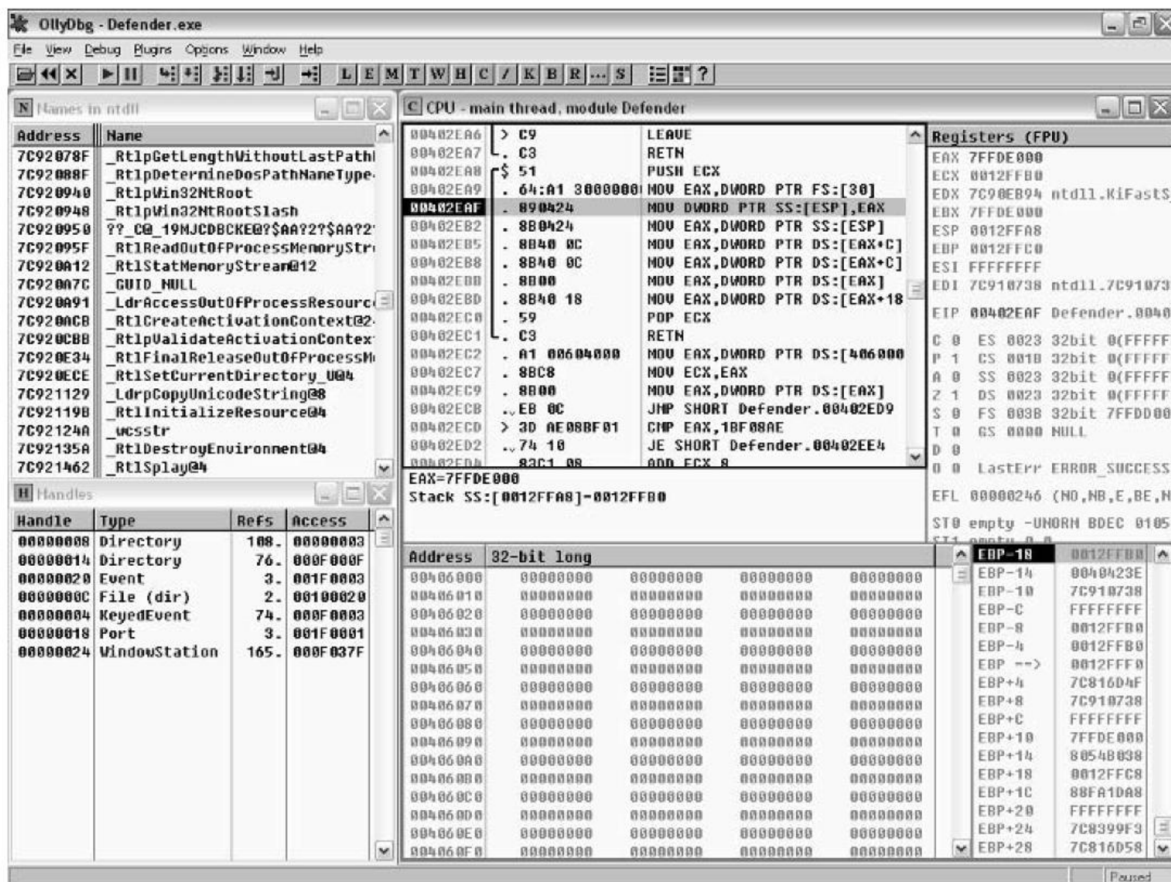
Ollydbg je x86 debugger, na analýzu binárneho kódu, čo je užitočné, keď zdrojový kód nie je k dispozícii. Ollydbg sleduje registre, rozoznáva procedúry, API volania, prepínače, tabuľky, konštanty a reťazce, rovnako ako lokalizuje rutiny z objektových súborov a knižníc. Verzia 1.10 je posledná verzia 1.x. Verzia 2.0 bola uvoľnená len nedávno, a v tejto verzii bol OllyDbg prepísaný od základu.

OllyDbg, napísaný Olehom Yuschukom, je pravdepodobne najlepší debugger pre reverzných inžinierov (aj keď výber v tejto oblasti je pomerne malý).

Krásna OllyDbg je v tom, že od základu bol navrhnutý ako nástroj pre reverzných inžinierov, a ako taký má veľmi dobrý vstavaný disassembler. Pomerne veľa začiatočníkov v reverznom inžinierstve začína s komplexnými nástrojmi, ako sú Numega SoftICE. Faktom je, že ak nepotrebujeme analyzovať kernel kód, alebo pozorovať systém globálne v rámci viacerých procesov, zvyčajne je pre nás OllyDbg viac než postačujúci.

Najväčšia sila OllyDbg je v jeho disasembleri, ktorý poskytuje veľmi výkonné prostriedky na analýzu kódu. OllyDbg analyzátor kódu umožňuje identifikovať slučky, bloky prepínačov, a ďalšie kľúčové štruktúry kódu. Ukazuje názvy parametrov pre všetky známe funkcie a rozhrania API, podporuje hľadanie krížových odkazov v kóde a dátach vo všetkých možných smeroch. V skutočnosti

by bolo spravodlivé povedať, že má OllyDbg má najlepšie disasemblovacie schopnosti zo všetkých dostupných debuggerov (okrem IDA Pro debugger).



Obr. 6.2 Typická obrazovka OllyDbg [5]

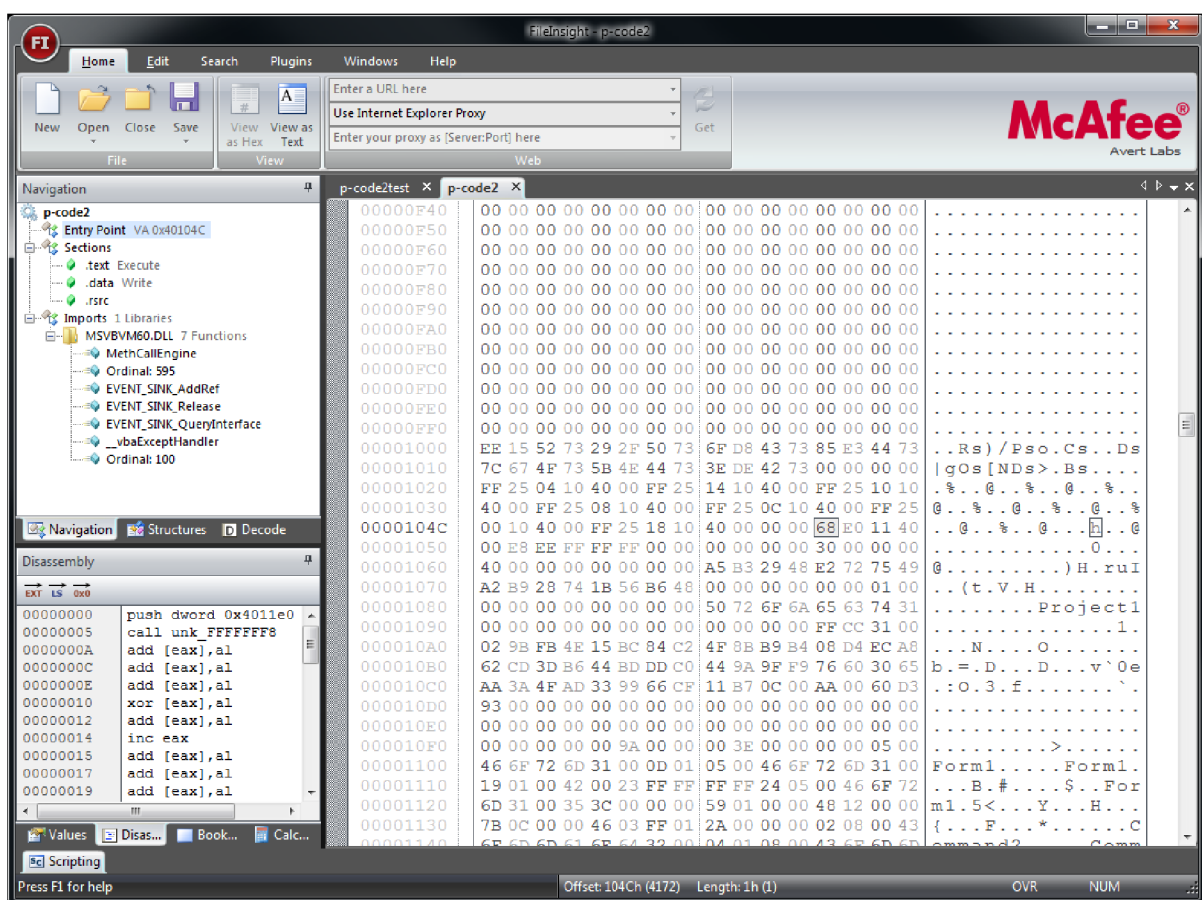
Okrem výkonnej funkcie disasemblingu, Ollydbg podporuje širokú škálu možností zobrazenia, vrátane zoznamu importov a exportov v moduloch, zobrazujúci zoznam okien a ďalších objektov, ukazujúci aktuálne reťazce výnimiek, pomocou importovaných knižníc (. lib súbory) pre pomenovanie funkcií, ktoré vznikli v týchto knižniciach, a iných.

OllyDbg tiež obsahuje vstavaný nástroj pre assembling a editáciu kódu, ktorý ho robí pre crackerov ešte obľúbenejším nástrojom. Je možné editovať kód assembleru na hocijakom mieste v programe a potom zmeny uložiť späť do spustiteľného súboru. Tiež je možné ukladať zoznam opráv pre konkrétny program a použiť niektoré alebo všetky z týchto opráv, zatiaľ čo tento program ladíme.

Obrázok 6.2 ukazuje typické OllyDbg obrazovky. Všimnite si zoznam mien NTDLL na ľavej strane. OllyDbg neukazuje len Importy a Exporty, ale aj interné názvy (ak sú k dispozícii symboly). Vľavo dole sa nachádza zoznam aktuálne otvorených deskriptorov v procese. [5]

6.3 McAfee FileInsight

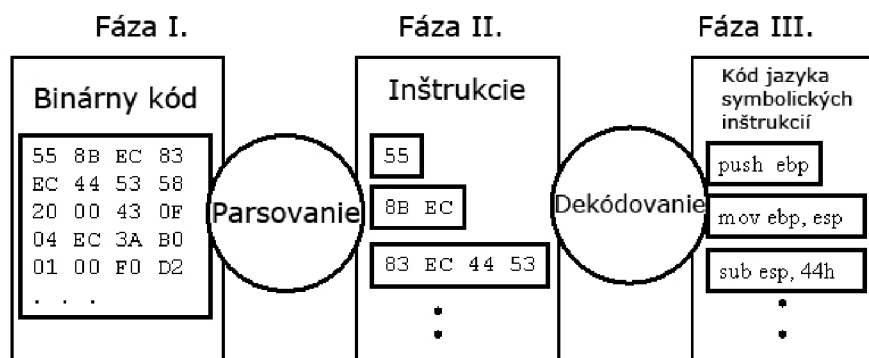
McAfee FileInsight, vyvíjaný McAfee Labs je prostredie pre analýzu súborov a web stránok. Poskytuje veľké množstvo zásuvných modulov a pomôcok pri analýze súborov, ktoré je možné veľmi jednoducho rozšíriť pomocou zásuvných modulov písaných v skriptovacom jazyku Python. Je to taktiež veľmi dobre navrhnutý HEX editor, ktorý umožňuje kvalitnú analýzu binárnych súborov. Z PE súborov dokáže zobrazit' veľké množstvo užitočných informácií ako sú vstupný bod programu, informácie o sekciách, importy. Rovnako dokáže disassemblovať binárny kód na inštrukcie assembleru. Tento nástroj bol veľmi užitočný pri analýze PE súborov a veľkou mierou mi pomohol pri pochopení ich štruktúry.



Obr. 6.3 Typická obrazovka McAfee FileInsight

7 Návrh disassembleru

Disassembling binárneho kódu do jazyka symbolických inštrukcií môže znieť veľmi zložito. Ak sa pozrieme do manuálu pre inštrukcie od Intelu zistíme, že to aj zložité je. Pri návrhu disassemblera nám pomôžu rôzne existujúce nástroje, ako je napríklad „dumpbin.exe“, čo je nástroj obsiahnutý v Microsoft Visual C++ nástrojoch, ale aj už spomenuté disassemblery IDA Pro a OllyDbg. Disassembling môžeme rozdeliť do troch hlavných fáz: [18]



Obr. 7.1 Fázy disasemblingu [18]

Toto by platilo ak by išlo o spustiteľný súbor C++. Avšak pri súboroch Visual Basic je situácia iná a tento spôsob sa dá využiť len sčasti. Spustiteľné súbory kompilované Visual Basicom hneď na začiatku začínajú používať funkcie obsiahnuté v DLL súbore MSVBVM60.DLL, konkrétne funkcia ThunRTMain, viď obrázok 7.2. Tento súbor preberá beh celého programu a pristupuje do spustiteľného súboru, kde sa pozerá aké ďalšie funkcie majú nasledovať.

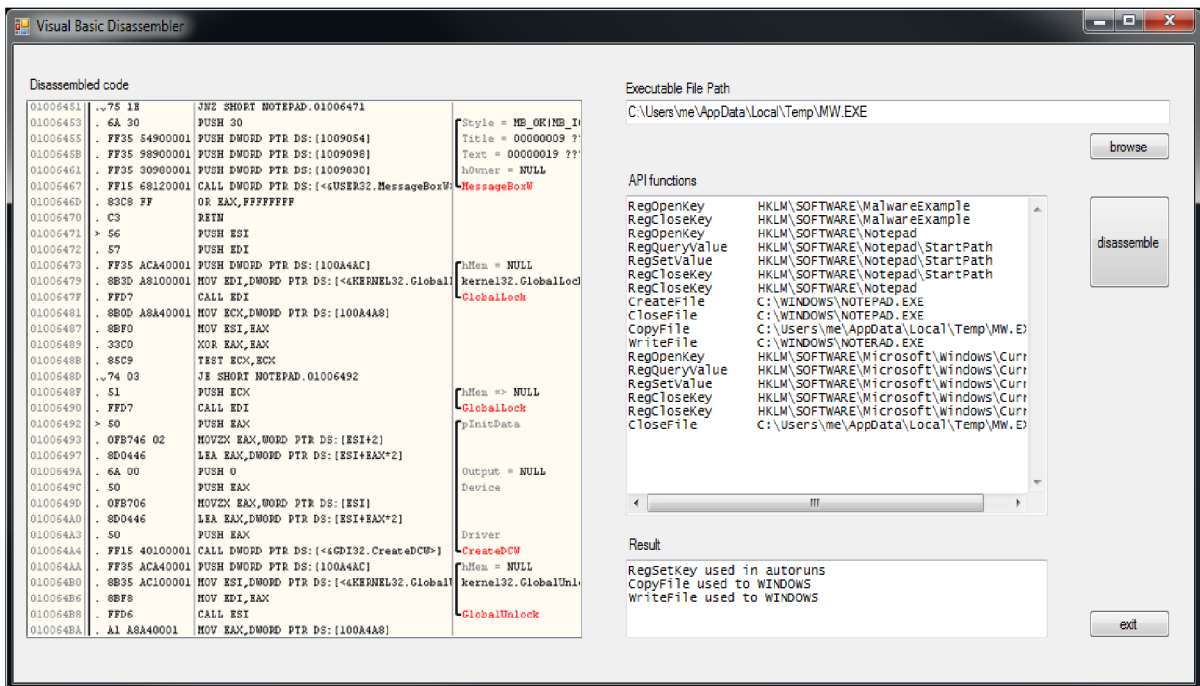
```
.text:0040114C  
.text:0040114C  
.text:0040114C start:  
* .text:0040114C  
* .text:00401151  
public start  
push offset dword_4012B4  
call ThunRTMain
```

Obr. 7.2 Začiatkový bod spustiteľného súboru Visual Basic

Z tohto dôvodu bude implementácia disassembleru pomerne zložitá. Vyžaduje si dôkladnú analýzu hlavného Visual Basic DLL súboru MSVBVM60.DLL. Túto analýzu plánujem vykonávať pomocou disassemblerov IDA Pro a OllyDbg. V prostredí Microsoft Visual Basic 6.0 si budem vytvárať postupne rôzne programy, ktoré budú využívať rozličné funkcie tohto jazyka a budem ich kompilovať s rôznymi možnosťami, ktoré kompilátor Visual Basic ponúka. Tieto spustiteľné súbory budem postupne analyzovať spolu s funkciami, ktoré poskytuje DLL súbor MSVBVM60.DLL. Táto

úloha sa môže zdať pomerne jednoduchá a jasná, ale opak je pravdou, pretože takýchto funkcií je veľmi veľa a k binárnym kódom Visual Basic 6.0 a nižšie neexistuje žiadna dokumentácia, ktorá by tieto dáta obsahovala. Čiže neostáva žiadna iná možnosť ako urobiť databázu všetkých používaných funkcií spolu s ich kódmi v binárnych súboroch Visual Basic. Po úspešnom dokončení tohto kroku, využijem tieto informácie pri následnej implementácii Disassembleru.

Disassembler plánujem implementovať v prostredí Microsoft Visual Studio 2010 v programovacom jazyku C#. Ukážku návrhu grafického rozhrania vidíme na obrázku 7.3. Hotová aplikácia by mala obsahovať okno s kódom v jazyku symbolických inštrukcií, okno so všetkými programom použitými Windows API funkciami a okno so štatistikou použitia Windows API funkcií, ktoré sú štandardne využívané hlavne užívateľovi škodlivými súbormi.



Obr. 7.3 Návrh grafického rozhrania Disassembleru Visual Basic

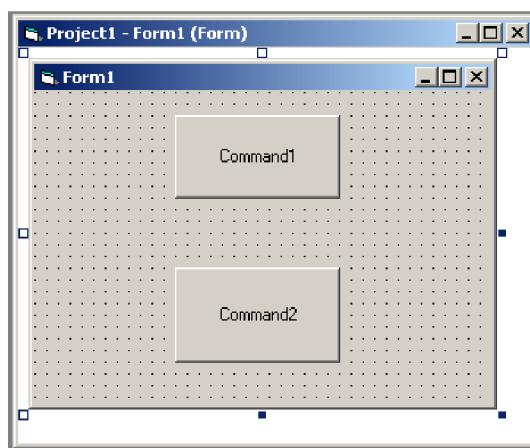
8 Analýza a implementácia

Najdôležitejšou časťou tejto diplomovej práce bola dôkladná analýza binárneho formátu .exe súborov Visual Basic 6 a následne schematické načrtnutie zloženia týchto súborov. Táto časť by bola veľmi jednoduchá, avšak opak je pravdou. Vývojári vo firme Microsoft po sebe nezanechali žiadne dokumenty ohľadom zloženia binárnych súborov Visual Basic 6. Respektíve ak nejaké zanechali, tak tieto dokumenty nie sú verejnosti prístupné a ani po dlhom pátraní sa k žiadnym dokumentom ohľadom zloženia binárnych súborov Visual Basic 6 nepodarilo dopátrať. Moja analýza teda spočívala v dlhodobom pozorovaní binárnych súborov Visual Basic 6 v hex editoroch, disassembleroch, nástrojoch na analýzu prístupu v súboroch a následnom premýšľaní, čo môžu jednotlivé položky, tabuľky, ofsety, pointre znamenať. Na tieto účely boli použité mnou vytvorené jednoduché aj zložitejšie programy skompilované v prostredí Visual Basic 6. Pri ich kompilovaní boli volené rozdielne nastavenia kompilátora:

- Compile to P-Code
- Compile to Native Code
 - Optimize for Fast Code
 - Optimize for Small Code
 - No Optimization

Po dlhšej dobe boli nájdené informácie, ktoré pomohli približne načrtnúť štruktúry binárnych súborov Visual Basic 6.

V nasledujúcom texte je presne popísaný postup prístupu k jednotlivým položkám binárnych súborov Visual Basic 6. Ako príklad je uvedený jednoduchý binárny súbor hello.exe skompilovaný pod Visual Basic 6 s nastavením kompilátora Compile to P-Code. Tento súbor obsahuje 1 formulár a 2 tlačidlá, ktoré po stlačení vypisujú text „Hello, World!“ a „Ahoj Svet!“. Na obrázku 8.1 vidíme ako tento program vyzerá v dizajnéri Visual Basic.



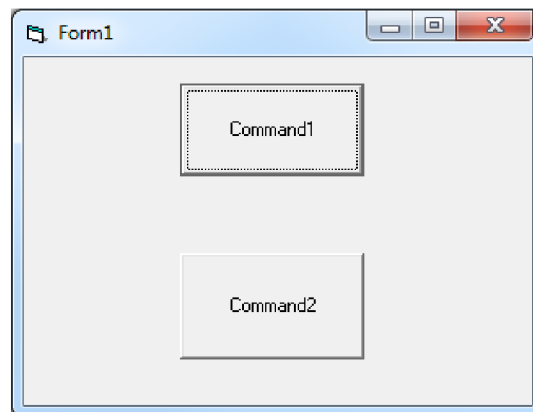
Obr. 8.1 Program hello.exe v dizajnéri Visual Basic

Jeho zdrojový kód vyzerá nasledovne:

```
1. Private Sub Command1_Click()  
2. MsgBox "Hello, World!"  
3. End Sub  
4.  
5. Private Sub Command2_Click()  
6. MsgBox "Ahoj Svet!"  
7. End Sub
```

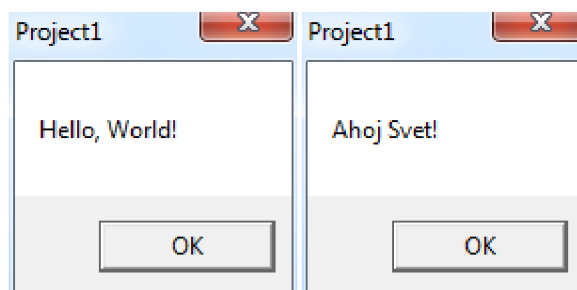
Zdrojový kód nášho referenčného Visual Basic programu

Ako vyzerá tento program po spustení skompilovaného .exe súboru vidíme na obrázku 8.2.



Obr. 8.2 Program hello.exe po spustení

Po stlačení jednotlivých tlačidiel sú zobrazené MsgBoxy zobrazené na obrázku 8.3.



Obr 8.3 MsgBoxy, zobrazené po kliknutí na tlačidlá v hello.exe

8.1 Získanie adresy Entry Point a iných základných informácií o PE súbore

Ako sme si vysvetlili v kapitole 2. Každý binárny súbor spustiteľný pod Operačným systémom Windows obsahuje MS-DOS hlavičku identifikovanú reťazcom MZ na adrese 00000000, na konci ktorej na adrese 0000003C sa nachádza adresa PE hlavičky. Adresa PE hlavičky je teda vždy 60 bajtov (0x3C) od začiatku MS-DOS hlavičky.

00000000	4D 5A 90 00 03 00 00 00	04 00 00 00 FF FF 00 00	MZ
00000010	B8 00 00 00 00 00 00 00	40 00 00 00 00 00 00 00@.....
00000020	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00
0000003C	00 00 00 00 00 00 00 00	00 00 00 00 B8 00 00 00

Obr. 8.4 Adresa PE hlavičky = 0x000000B8

000000B8	00 00 00 00 00 00 00 00	50 45 00 00 4C 01 03 00PE..L...
000000C0	97 D2 C7 4D 00 00 00 00	00 00 00 00 E0 00 0F 01	...M.....
000000D0	0B 01 06 00 00 10 00 00	00 20 00 00 00 00 00 00
000000E0	4C 10 00 00 00 10 00 00	00 20 00 00 00 00 40 00	L.....@.
000000F0	00 10 00 00 00 10 00 00	04 00 00 00 01 00 00 00
00000100	04 00 00 00 00 00 00 00	00 40 00 00 00 10 00 00@.....
00000110	CB F1 00 00 02 00 00 00	00 00 10 00 00 10 00 00
00000120	00 00 10 00 00 10 00 00	00 00 00 00 10 00 00 00

Obr. 8.5 Začiatok PE hlavičky na adrese 0x000000B8

000000BF	00 00 00 00 00 00 00 00	50 45 00 00 4C 01 03 00PE..L..
000000C0	97 D2 C7 4D 00 00 00 00	00 00 00 00 E0 00 0F 01	...M.....
000000D0	0B 01 06 00 00 10 00 00	00 20 00 00 00 00 00 00
000000E0	4C 10 00 00 00 10 00 00	00 20 00 00 00 00 40 00	L.....@.
000000F0	00 10 00 00 00 10 00 00	04 00 00 00 01 00 00 00
00000100	04 00 00 00 00 00 00 00	00 40 00 00 00 10 00 00@.....
00000110	CB F1 00 00 02 00 00 00	00 00 10 00 00 10 00 00
00000120	00 00 10 00 00 10 00 00	00 00 00 00 10 00 00 00

Obr. 8.6 Počet sekcií = 0x0003 = 3

000000B0	00 00 00 00 00 00 00 00	50 45 00 00 4C 01 03 00PE..L...
000000C0	97 D2 C7 4D 00 00 00 00	00 00 00 00 E0 00 0F 01	...M.....
000000D0	0B 01 06 00 00 10 00 00	00 20 00 00 00 00 00 00
000000E0	4C 10 00 00	00 10 00 00 00 20 00 00 00 00 40 00	L.....@.
000000F0	00 10 00 00 00 10 00 00	04 00 00 00 01 00 00 00
00000100	04 00 00 00 00 00 00 00	00 40 00 00 00 10 00 00@.....
00000110	CB F1 00 00 02 00 00 00	00 00 10 00 00 10 00 00
00000120	00 00 10 00 00 10 00 00	00 00 00 00 10 00 00 00

Obr. 8.7 Adresa Entry Point = 0x0000104C

000000B0	00 00 00 00 00 00 00 00	50 45 00 00 4C 01 03 00PE..L...
000000C0	97 D2 C7 4D 00 00 00 00	00 00 00 00 E0 00 0F 01	...M.....
000000D0	0B 01 06 00 00 10 00 00	00 20 00 00 00 00 00 00
000000EC	4C 10 00 00 00 10 00 00	00 20 00 00 00 00 40 00	L.....@.
000000F0	00 10 00 00 00 10 00 00	04 00 00 00 01 00 00 00
00000100	04 00 00 00 00 00 00 00	00 40 00 00 00 10 00 00@.....
00000110	CB F1 00 00 02 00 00 00	00 00 10 00 00 10 00 00
00000120	00 00 10 00 00 10 00 00	00 00 00 00 10 00 00 00

Obr. 8.8 ImageBase = 0x00400000

000000B0	00 00 00 00 00 00 00 00	50 45 00 00 4C 01 03 00PE..L...
000000C0	97 D2 C7 4D 00 00 00 00	00 00 00 00 E0 00 0F 01	...M.....
000000D0	0B 01 06 00 00 10 00 00	00 20 00 00 00 00 00 00
000000E0	4C 10 00 00 00 10 00 00	00 20 00 00 00 00 40 00	L.....@.
000000F0	00 10 00 00 00 10 00 00	04 00 00 00 01 00 00 00
00000100	04 00 00 00 00 00 00 00	00 40 00 00 00 10 00 00@.....
00000110	CB F1 00 00 02 00 00 00	00 00 10 00 00 10 00 00
0000012F	00 00 10 00 00 10 00 00	00 00 00 00 10 00 00 00

Obr. 8.9 NumberOfRvaAndSizes = 0x00000010 = 16

Avšak adresa Entry Point, ktorá sa tu nachádza je relatívna adresa platná po zavedení súboru do pamäte. Ak chceme poznať skutočnú adresu Entry Point v binárnom súbore potrebujeme poznať veľkosť ImageBase a počet NumberOfRvaAndSizes.

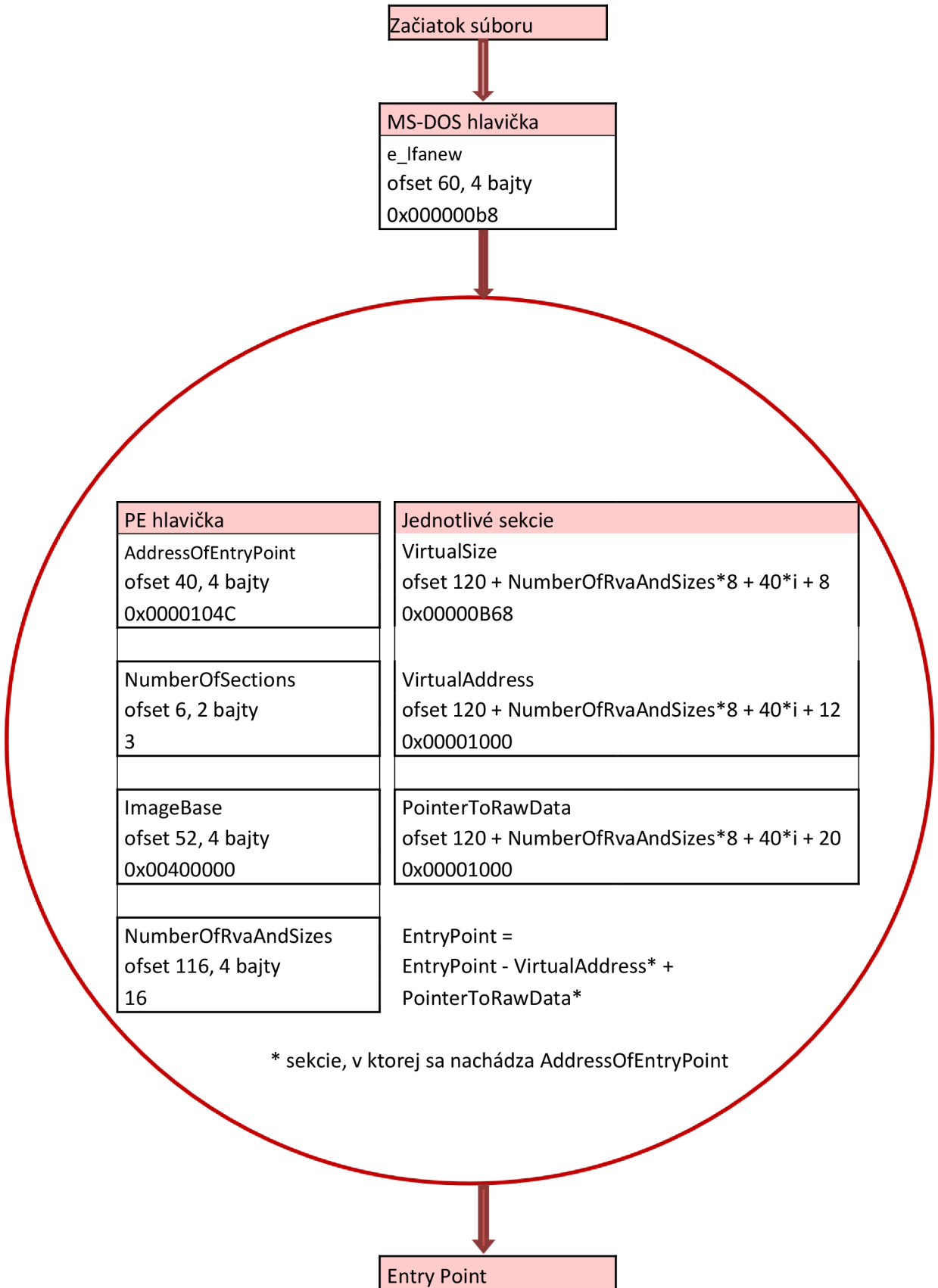
Skutočnú adresu Entry Point zistíme nasledovne:

1. Zistíme Adresu Entry point = 0x0000104C
2. O každej sekcii si zistíme VirtualSize, VirtualAddress a PointerToRawData
 - a. To zistíme tak, že od začiatku PE hlavičky preskočíme 120 bajtov (to je veľkosť PE hlavičky + veľkosť nepovinnnej hlavičky)
 - b. Ďalej potrebujeme preskočiť dátové štruktúry. Ich počet je udaný hodnotou NumberOfRvaAndSizes a každá dátová štruktúra má veľkosť 8 bajtov.
 - c. V tomto okamihu sme na začiatku informácií o prvej sekcii. Prvých 8 bajtov je názov sekcie. Tie preskočíme a dostaneme sa na informáciu VirtualSize sekcie, ktorá má 4 bajty.
 - d. Ďalšie 4 bajty obsahujú informáciu o VirtualAddress sekcie.
 - e. Preskočíme ďalšie 4 bajty (tie obsahujú informáciu o SizeOfRawData, ktorá je pre nás v tomto prípade nepodstatná).
 - f. Ďalšie 4 bajty obsahujú informáciu o PointerToRawData.
 - g. Veľkosť informácií o jednej sekcii je 40 bajtov. Ak sa teda chceme dostať k informáciám o 2. sekcii potrebujeme preskočiť od PE hlavičky: 120 + NumberOfRvaAndSizes * 8 + 40 bajtov. Informácie o 2. sekcii dostaneme ak budeme pokračovať bodom c)
 - h. Toto opakujeme podľa počtu sekcií, počet sekcií poznáme z PE hlavičky
3. Potrebujeme zistiť, v ktorej sekcii sa Entry Point nachádza. To zistíme tak, že Adresa Entry Point je menšia ako VirtualAddress + VirtualSize sekcie a zároveň väčšia ako VirtualAddress sekcie.

4. Ak vieme, v ktorej sekcii sa Entry Point nachádza, vypočítame ho jednoducho:
$$\text{EntryPoint} = \text{EntryPoint} - \text{VirtualAddress} \text{ sekcie} + \text{PointerToRawData} \text{ sekcie}$$

Názov štruktúry
Názov položky danej štruktúry
Ofset položky v štruktúre , dĺžka položky
Konkrétna adresa odkazovanej štruktúry v súbore "p-code2.exe"

Tabuľka 8.1 Formát uvedených informácií na obrázku 8.11



Obr. 8.11 Získanie Entry Point

00000000	4D 5A 90 00 03 00 00 00	04 00 00 00 FF FF 00 00	MZ.....	00000000 - 0000003F
00000010	B8 00 00 00 00 00 00 00	40 00 00 00 00 00 00 00@.....	MS-DOS hlavička
00000020	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	na adrese 0000003A je adresa
00000030	00 00 00 00 00 00 00 00	00 00 00 00 B8 00 00 00	PE hlavičky
00000040	0E 1F BA 0E 00 B4 09 CD	21 B8 01 4C CD 21 54 68!..L.!Th	DOS STUB
00000050	69 73 20 70 72 6F 67 72	61 6D 20 63 61 6E 6E 6F	is program canno	
00000060	74 20 62 65 20 72 75 6E	20 69 6E 20 44 4F 53 20	t be run in DOS	
00000070	6D 6F 64 65 2E 0D 0D 0A	24 00 00 00 00 00 00 00	mode....\$......	Rich hlavička
00000080	B7 12 07 DB F3 73 69 88	F3 73 69 88 F3 73 69 88si..si..si.	
00000090	1A 6C 64 88 F2 73 69 88	52 69 63 68 F3 73 69 88	.ld..si.Rich.si.	
000000A0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000000B8	00 00 00 00 00 00 00 00	50 45 00 00 4C 01 03 00PE..L..	000000B8 - 0000012F
000000C0	97 D2 C7 4D 00 00 00 00	00 00 00 00 E0 00 0F 01	...M.....	PE hlavička
000000D0	0B 01 06 00 00 10 00 00	00 20 00 00 00 00 00 00	
000000E0	4C 10 00 00 00 10 00 00	00 20 00 00 00 00 40 00	L.....@.	000000E0 - Adresa Entry Point =
000000F0	00 10 00 00 00 10 00 00	04 00 00 00 01 00 00 00	0000104C
00000100	04 00 00 00 00 00 00 00	00 40 00 00 00 10 00 00@.....	
00000110	CB F1 00 00 02 00 00 00	00 00 10 00 00 10 00 00	
0000012F	00 00 10 00 00 10 00 00	00 00 00 00 10 00 00 00	
00000130	00 00 00 00 00 00 00 00	A4 1A 00 00 28 00 00 00(.....	00000130 - 000001AF
00000140	00 30 00 00 BC 08 00 00	00 00 00 00 00 00 00 00	.0.....	Dátové štruktúry - každá má
00000150	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	veľkosť 8 bajtov ich počet je daný
00000160	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	poslednými 4 bajtmi nepovinnnej
00000170	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	hlavičky (v tomto prípade na
00000180	00 00 00 00 00 00 00 00	28 02 00 00 20 00 00 00(.....	adrese 0000012A je 0x00000010
00000190	00 10 00 00 20 00 00 00	00 00 00 00 00 00 00 00	= 16) 16x8=128 bajtov
000001AF	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
000001B0	2E 74 65 78 74 00 00 00	68 0B 00 00 00 10 00 00	.text...h.....	000001B0 - 000001D7
000001C0	00 10 00 00 00 10 00 00	00 00 00 00 00 00 00 00	1. sekcia .text
000001D8	00 00 00 00 20 00 00 60	2E 64 61 74 61 00 00 00data.....	000001D8 - 000001FF
000001E0	E0 09 00 00 00 20 00 00	00 00 00 00 00 00 00 00	2. sekcia .data
000001FF	00 00 00 00 00 00 00 00	00 00 00 00 40 00 00 C0@.....	
00000200	2E 72 73 72 63 00 00 00	BC 08 00 00 00 30 00 00	.rsrc.....0..	00000200 - 00000227
00000210	00 10 00 00 00 20 00 00	00 00 00 00 00 00 00 00	3. sekcia .rsrc
00000227	00 00 00 00 40 00 00 40	50 97 10 41 10 00 00 00@..@P..A....	

Obr. 8.12 Prehľad informácií o PE súbore

Môže sa zdať, že Adresa Entry Point sa nezmenila, avšak to len preto, že v tomto konkrétnom prípade sa VirtualAddress sekcie a PointerToRawData sekcie rovnajú. Toto nemusí byť pravidlom a v mnohých prípadoch sa tieto hodnoty líšia. [32]

8.2 Získanie Importov

Najskôr by sme si mali ozrejmiť, čo to vlastne importy sú. Funkcia Import je funkcia, ktorá nie je obsiahnutá vo volajúcom module, ale je týmto modulom volaná, odtiaľ pochádza názov „Import“. Importy sa obyčajne nachádzajú v jednej alebo aj viacerých DLL knižniciach. Vo volajúcom module je obsiahnutá iba informácia o funkciách. Tieto informácie obsahujú názvy funkcií a názvy DLL knižníc, v ktorých sa jednotlivé funkcie nachádzajú. [15]

1. Od adresy PE hlavičky preskočíme 120 bajtov (PE hlavička a nepovinná hlavička)
2. Teraz sa nachádzame na adrese, kde sú definované informácie o dátových štruktúrach.
3. Preskočíme ďalších 8 bajtov, tie definujú Exporty.
4. Prvé 4 bajty definujú ImportVirtualAddress, ďalšie 4 definujú veľkosť importov, vid' obrázok 8.12. [25]

```
00000138 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | A4 1A 00 00 28 00 00 00 | ..... (.....)
```

Obr. 8.13 Import Directory

5. 00001AA4 je adresa, kde sa nachádza tabuľka importov vid' obrázok 8.12.

Tabuľka importov je vlastne pole IMAGE_IMPORT_DESCRIPTOR štruktúr, ktoré sú definované nasledovne:

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

Prvá položka je zjednotenie OriginalFirstThunk, čo môžeme považovať za charakteristiku. Tento člen obsahuje adresu polí IMAGE_THUNK_DATA štruktúr.

IMAGE_THUNK_DATA je ukazateľ na štruktúru IMAGE_IMPORT_BY_NAME, nie je to samotná štruktúra IMAGE_IMPORT_BY_NAME.

Môžeme sa na to pozrieť takto: Existuje viacero štruktúr IMAGE_IMPORT_BY_NAME. Získame adresy všetkých týchto štruktúr a uložíme ich do poľa IMAGE_THUNK_DATA ukončeného 0. Adresu poľa IMAGE_THUNK_DATA uložíme ako OriginalFirstThunk.

Štruktúra IMAGE_IMPORT_BY_NAME obsahuje informácie o importovanej funkcii. Je definovaná nasledovne:

```
IMAGE_IMPORT_BY_NAME STRUCT
    Hint dw ?
    Name2 db ?
IMAGE_IMPORT_BY_NAME ENDS
```

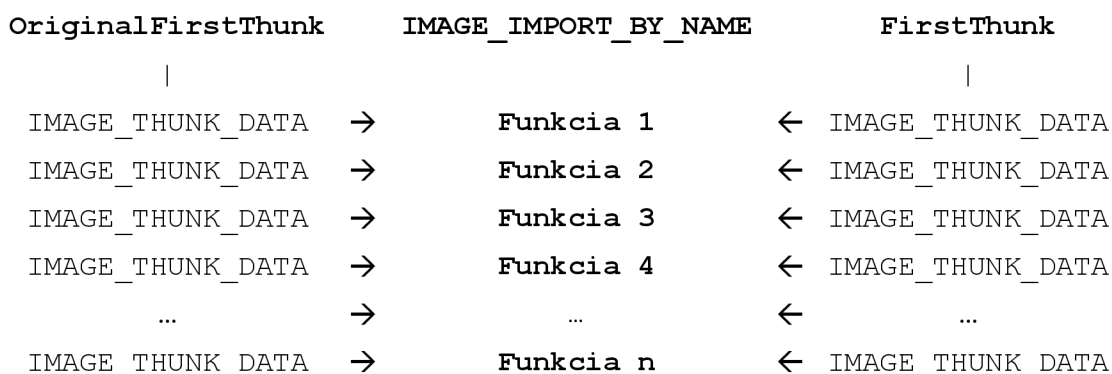
Premenná Hint obsahuje index do tabuľky exportov DLL knižnice, kde sa funkcia nachádza. Býva využívaný PE zavádzačom, aby mohol urýchlene nájsť funkciu v tabuľke exportov DLL knižnice. Táto hodnota nie je povinná a často býva nastavená na 0.

Premenná Name2 obsahuje názov importovanej funkcie. Je zložený z ASCII reťazca variabilnej dĺžky.

Premenná Name1 obsahuje adresu názvu DLL knižnice, v skratke je to pointer na názov DLL knižnice. Tento názov je ASCII reťazec variabilnej dĺžky.

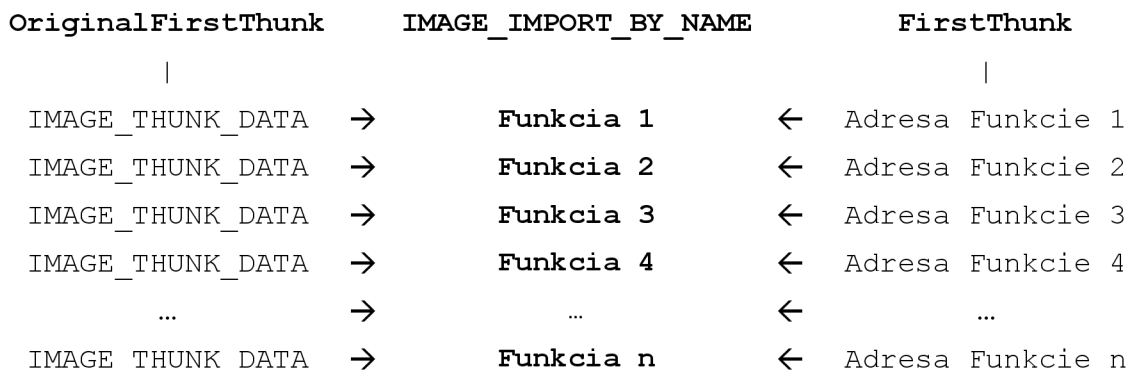
FirstThunk je veľmi podobný OriginalFirstThunk, obsahuje adresu polí IMAGE_THUNK_DATA štruktúr (je to však odlišné pole ako je definované položkou OriginalFirstThunk).

Pozrime sa na to takto: Existuje viacero štruktúr IMAGE_IMPORT_BY_NAME. Vytvoríme dva polia a uložíme do nich adresy štruktúr IMAGE_IMPORT_BY_NAME. Čiže obidva polia obsahujú rovnaké adresy, tzv. duplikáty. Adresu prvého poľa uložíme ako OriginalFirstThunk a adresu druhého poľa uložíme ako FirstThunk.



Počet položiek v poli na ktoré ukazuje OriginalFirstThunk a FirstThunk závisí na funkciách, ktoré PE súbor importuje z DLL knižnice. Napríklad: ak PE súbor importuje 10 funkcie z kernel32.dll, Name1 v štruktúre IMAGE_IMPORT_DESCRIPTOR obsahuje adresu reťazca „kernel32.dll“ a OriginalFirstThunk a FirstThunk ukazuje na polia s 10 položkami.

Ďalšia otázka môže byť: prečo potrebujeme dva úplne rovnaké polia? Aby sme túto otázku zodpovedali potrebujeme vedieť, že keď je PE súbor zavedený do pamäte, PE zavádzač sa pozrie na IMAGE_THUNK_DATA a IMAGE_IMPORT_BY_NAME a zistí adresy importovaných funkcií. Potom prepíše IMAGE_THUNK_DATA v poli, na ktoré ukazuje FirstThunk skutočnými adresami týchto funkcií. [16] Čiže, keď je PE súbor pripravený PE zavádzačom na spustenie, štruktúra sa zmení na:



Na obrázku 8.14 vidíme štruktúru importov znázornenú priamo v binárnom súbore Visual Basic 6.

00001A70 | 00 00 00 00 5C FF 02 00 | 3C FF 02 00 1C FF 02 00 |\
00001A80 | FC FE 02 00 CC CC CC CC | CC CC CC CC CC CC CC CC |<
00001A90 | E9 E9 E9 E9 CC CC CC CC | CC CC CC CC CC CC CC CC |
00001AA0 | 9E 9E 9E 9E CC 1A 00 00 | FF FF FF FF FF FF FF FF |
00001AB0 | EC 1A 00 00 00 10 00 00 | 00 00 00 00 00 00 00 00 |
00001AC0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 FA 1A 00 00 |
00001AD0 | 53 02 00 80 0C 1B 00 00 | 20 1B 00 00 36 1B 00 00 |
00001AE0 | 52 1B 00 00 62 00 00 80 | 00 00 00 00 4D 53 56 42 | R...d...MSVB
00001AF0 | 56 4D 36 30 5F 44 4C 4C | 00 00 00 00 4D 65 74 68 | VM60.DLL...Meth
00001B00 | 43 61 6C 6C 45 6E 87 49 | 6E 65 00 00 00 00 45 56 | CallEngine...EV
00001B10 | 45 4E 54 5F 53 49 4E 4B | 5F 41 64 64 52 65 66 00 | ENT_SINK_AddRef.
00001B20 | 00 00 45 56 45 4E 54 5F | 53 49 4E 4B 5F 52 65 6C | ..EVENT_SINK_Rel
00001B30 | 65 61 73 65 00 00 00 00 | 45 56 45 4E 54 5F 53 49 | ease...EVENT_SI
00001B40 | 4E 4B 5F 51 75 65 72 79 | 49 6E 74 65 72 66 61 63 | NK_QueryInterfac
00001B50 | 65 00 00 00 5F 5F 76 62 | 61 45 78 63 65 70 74 48 | e..._vbaExceptH
00001B60 | 61 6E 64 6C 65 72 00 00 | 00 00 00 00 00 00 00 00 | andler.....
00001B70 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
00000FF0 | 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 |
00001000 | EE 15 52 73 29 2F 50 73 | 6F D8 43 73 85 E3 44 73 | ..Rs) / Pso.Cs.Ds
00001010 | 7C 67 4F 73 5B 4E 44 73 | 3E DE 42 73 00 00 00 00 | |gOs[NDs>.Bs...
00001020 | FF 25 04 10 40 00 FF 25 | 14 10 40 00 FF 25 10 10 | |%.%.%.%.%.%.%.%

00001AA4 + 20 bajtov =
IMAGE_IMPORT_DESCRIPTOR
00001AA4 OriginalFirstThunk
00001AB0 Name1 00001AB4 FirstThunk
00001ACC začiatok tabuľky štruktúr IMAGE_THUNK_DATA, na ktoré ukazuje OriginalFirstThunk
00001AEC názov DLL knižnice
00001AFA IMAGE_IMPORT_BY_NAME
prvé 2 bajty je Hint (pre nás nepodstatné) nasleduje názov importovanej funkcie

00001000 začiatok tabuľky štruktúr IMAGE_THUNK_DATA, na ktoré ukazujú FirstThunk

Obr. 8.14 Štruktúra importov v binárnom súbore VB6

```
Imports:
dll[0] MSVBVM60.DLL
function[0] MethCallEngine
function[1] 80000253
function[2] EVENT_SINK_AddRef
function[3] EVENT_SINK_Release
function[4] EVENT_SINK_QueryInterface
function[5] __vbaExceptionHandler
function[6] 80000064
```

V tabuľke importov môže byť väčší počet štruktúr IMAGE_IMPORT_DESCRIPTOR, počet týchto štruktúr závisí na počte DLL knižníc, z ktorých importujeme jednotlivé funkcie. Počet IMAGE_IMPORT_DESCRIPTOR štruktúr vieme zistiť jednoducho z dátovej štruktúry Imports, kde je definovaná ich veľkosť v bajtoch. V našom prípade je táto veľkosť 0x00000028 = 40 bajtov. Každá

štruktúra IMAGE_IMPORT_DESCRIPTOR má 20 bajtov a v celkovej veľkosti je zahrnutá aj ukončovacia štruktúra IMAGE_IMPORT_DESCRIPTOR so samými nulami. Čiže počet IMAGE_IMPORT_DESCRIPTOR štruktúr = (veľkosť import tabuľky - 20) / 20, čo je v našom prípade 1. [19][20]

8.3 Informácie o súbore Visual Basic 6

Informácie o súbore skompilovanom vo Visual Basic 6 sa nachádzajú v rôznych štruktúrach, hlavičkách a tabuľkách. K jednotlivým informáciám je možné sa dostať z hlavičky Visual Basic súboru.

8.3.1 Visual Basic hlavička

K hlavičke Visual Basic sa dostaneme nasledovne:

1. Z predchádzajúcich informácií vieme adresu Entry Point, ktorá je v našom prípade 0000104C
2. Na adrese Entry Point sa nachádza inštrukcia push Virtuálna Adresa Visual Basic hlavičky (68 E0 11 40 00). Virtuálna Adresa Visual Basic hlavičky je v našom prípade 004011E0. Skutočnú adresu získame ak od tejto hodnoty odpočítame ImageBase.
004011E0 - 00400000 = 11E0

```

0000104C  | 00 10 40 00 FF 25 18 10 | 40 00 00 00 68 E0 11 40 | . . @ . . % . . @ . . h . . @ |
00001050  | 00 E8 EE FF FF FF 00 00 | 00 00 00 00 30 00 00 00 | . . . . . . . . 0 . . . . . |

```

Obr. 8.15 Adresa Visual Basic hlavičky pri Entry Pointe

Visual Basic hlavička je hlavný deskriptor ľubovoľného súboru skompilovaného pod Visual Basic. Táto štruktúra obsahuje ofsety, ukazatele a väzby na ostatné dôležité štruktúry v súbore. Taktiež poskytuje dôležité informácie o programe a o použítom jazyku.

Visual Basic hlavička je štruktúra o veľkosti 0x68 = 108 bajtov. V tabuľke 8.2 je znázornená jej štruktúra.

Offset	Názov	Popis
0x0	szVbMagic	"VB5!" reťazec
0x4	wRuntimeBuild	verzia VB6 Runtime
0x6	szLangDll	jazyková DLL knižnica
0x14	szSecLangDll	druhá jazyková DLL knižnica
0x22	wRuntimeRevision	interná revízia Runtime
0x24	dwLCID	LCID jazykovej DLL knižnice
0x28	dwSecLCID	LCID druhej jazykovej DLL knižnice
0x2C	lpSubMain	ukazateľ na sub hlavný kód
0x30	lpProjectData	ukazateľ na sub Project Data
0x34	fMdlIntCtls	VB príznaky pre ID < 32
0x38	fMdlIntCtls2	VB príznaky pre ID > 32
0x3C	dwThreadFlags	mód vlákien
0x40	dwThreadCount	počet podporovaných vlákien
0x44	wFormCount	počet foriem
0x46	wExternalCount	počet externých prvkov
0x48	dwThunkCount	počet thunk na vytvorenie
0x4C	lpGuiTable	ukazateľ na GUI tabuľku
0x50	lpExternalTable	ukazateľ na externú tabuľku
0x54	lpComRegisterData	ukazateľ na COM informácie
0x58	bSZProjectDescription	ofset na popis projektu
0x5C	bSZProjectExeName	ofset na názov EXE súboru
0x60	bSZProjectHelpFile	ofset na help súbor
0x64	bSZProjectName	ofset na meno projektu

Tabuľka 8.2 Visual hlavička (Visual Basic Header)

Ofset na popis projektu, ofset na názov súboru, ofset na help súbor a ofset na meno projektu sú ofsety od začiatku Visual Basic hlavičky.

000011E0	56 42 35 21 F0 1F 2A 00	00 00 00 00 00 00 00 00	VB5!...*
000011F0	00 00 00 00 7E 00 00 00	00 00 00 00 00 00 00 00	...~
00001200	00 00 0A 00 09 04 00 00	00 00 00 00 00 00 00 00	...
00001210	5C 14 40 00 10 F0 30 00	00 FF FF FF 08 00 00 00	\. @...0
00001220	01 00 00 00 01 00 00 00	E9 00 00 00 90 11 40 00	...@.
00001230	90 11 40 00 58 10 40 00	78 00 00 00 7E 00 00 00	..@.X.@.x...~
00001240	87 00 00 00 88 00 00 00	00 00 00 00 00 00 00 00	...
00001250	00 00 00 00 00 00 00 00	68 65 6C 6C 6F 00 50 72	...hello.Pr
00001260	6F 6A 65 63 74 31 00 00	50 72 6F 6A 65 63 74 31	oject1..Project1
00001270	00 00 00 00 00 00 00 00	24 20 40 00 6C 19 40 00	..\$ @.1.@.

Obr. 8.16 Visual Basic hlavička

Visual basic hlavička z nášho súboru obsahuje tieto informácie:

```

VB header:
szVbMagic          VB5!
wRuntimeBuild      21351FF0
szLangDll          2A
szSecLangDll       7E
wRuntimeRevision   A
dwLCID             409
dwSecLCID          0
lpSubMain          0
lpProjectData      40145C
fMdlIntCtls        30F010
fMdlIntCtls2       FFFFFFF00
dwThreadFlags      8
dwThreadCount      1
wFormCount         1
wExternalCount     0
dwThunkCount       E9
lpGuiTable         401190
lpExternalTable    401190
lpComRegisterData  401058
bSZProjectDescription 78
bSZProjectExeName  7E
bSZProjectHelpFile 87
bSZProjectName     88

ProjectDescription  hello
ProjectExeName     Project1
ProjectHelpFile    Project1
ProjectName        Project1
    
```

Hodnota	Názov	Popis
0x1	ApartmentModel	špecifikuje multi-threading Použitím apartment modelu
0x2	RequireLicense	špecifikuje validáciu licencie (len pre OCX)
0x4	Unattended	špecifikuje, že žiadne GUI prvky nemajú byť inicializované
0x8	SingleThreaded	špecifikuje single-threading
0x10	Retained	špecifikuje, že súbor má byť ponechný v pamäti

Tabuľka 8.3 Módy vlákien pre dwThreadFlags vo VB hlavičke

VB príznaky pre fMdlIntCtls vo VB hlavičke		
Control ID	Hodnota	Názov
0x0	0x1	PictureBox Object
0x1	0x2	Label Object
0x2	0x4	TextBox Object
0x3	0x8	Frame Object
0x4	0x10	CommandButton Object
0x5	0x20	CheckBox Object
0x6	0x40	OptionButton Object
0x7	0x80	ComboBox Object
0x8	0x100	ListBox Object
0x9	0x200	HScrollBar Object
0xA	0x400	VScrollBar Object
0xB	0x800	Timer Object
0xC	0x1000	Print Object
0xD	0x2000	Form Object
0xE	0x4000	Screen Object
0xF	0x8000	Clipboard Object
0x10	0x10000	Drive Object
0x11	0x20000	Dir Object
0x12	0x40000	FileListBox Object
0x13	0x80000	Menu Object
0x14	0x100000	MDIForm Object
0x15	0x200000	App Object
0x16	0x400000	Shape Object
0x17	0x800000	Line Object
0x18	0x1000000	Image Object
0x19	0x2000000	Nepodporované
0x1A	0x4000000	Nepodporované
0x1B	0x8000000	Nepodporované
0x1C	0x10000000	Nepodporované
0x1D	0x20000000	Nepodporované
0x1E	0x40000000	Nepodporované
0x1F	0x80000000	Nepodporované
2nd Flag Zone		
0x20	0x1	Nepodporované
0x21	0x2	Nepodporované
0x22	0x4	Nepodporované
0x23	0x8	Nepodporované
0x24	0x10	Nepodporované
0x25	0x20	DataQuery Object
0x26	0x40	OLE Object
0x27	0x80	Nepodporované
0x28	0x100	UserControl Object
0x29	0x200	PropertyPage Object
0x2A	0x400	Document Object
0x2B	0x800	Nepodporované

Tabuľka 8.4 VB príznaky pre fMdlIntCtls vo VB hlavičke

8.3.2 Informácie o projekte

Štruktúra informácie o projekte obsahuje užívateľské informácie o projekte ako aj veľmi dôležité informácie (ako napríklad ukazateľ na Tabuľku objektov).

Na štruktúru Informácie o projekte ukazuje ukazateľ `lpProjectData`, ktorý sa nachádza vo Visual Basic hlavičke.

Informácie o projekte je štruktúra o veľkosti `0x23C = 572` bajtov. V tabuľke 8.5 je znázomená jej štruktúra.

Offset	Názov	Popis
0x0	<code>dwVersion</code>	verzia 5.00 v hex (<code>0x01F4</code>)
0x4	<code>lpObjectTable</code>	ukazateľ na Tabuľku objektov
0x8	<code>dwNull</code>	nevyužívané
0xC	<code>lpCodeStart</code>	ukazateľ na začiatok kódu
0x10	<code>lpCodeEnd</code>	ukazateľ na koniec kódu
0x14	<code>dwDataSize</code>	veľkosť objektových štruktúr Visual Basic
0x18	<code>lpThreadSpace</code>	ukazateľ na ukazateľ na objekt vlákien
0x1C	<code>lpVbaSeh</code>	ukazateľ na VBA Exception Handler
0x20	<code>lpNativeCode</code>	ukazateľ na .DATA sekciu
0x24	<code>szPathInformation</code>	cesta pri kompilácii
0x234	<code>lpExternalTable</code>	ukazateľ na Externú tabuľku
0x238	<code>dwExternalCount</code>	počet objektov v Externej tabuľke

Tabuľka 8.5 Informácie o projekte (The Project Information)

Project info:

```
dwVersion          1F4
lpObjectTable      401274
dwNull             0
lpCodeStart        401A90
lpCodeEnd          401AA0
dwDataSize         9E0
lpThreadSpace      402000
lpVbaSeh           401026
lpNativeCode       0
szPathInformation  c:\Program Files\Microsoft Visual Studio\VB98\Project1.vbp
lpExternalTable2   401190
dwExternalCount    0
```

8.3.3 Sekundárne informácie o projekte

Štruktúra Sekundárne informácie o projekte obsahuje hlavne informácie potrebné pri kompilovaní projektu. Pomocou tejto štruktúry sa tiež dostávame na zoznam Foriem využitých v projekte.

Na štruktúru Sekundárne informácie o projekte ukazuje ukazateľ lpProjectInfo2, ktorý sa nachádza v Tabuľke objektov.

Sekundárne informácie o projekte je štruktúra o veľkosti 0x28 = 40 bajtov. V tabuľke 8.6 je znázomená jej štruktúra.

Offset	Názov	Popis
0x0	lpHeapLink	nevyužité, vždy 0
0x4	lpObjectTable	ukazateľ na Tabuľku objektov
0x8	dwReserved	nevyužité, vždy -1
0xC	dwUnused	nevyužité
0x10	lpObjectList	ukazateľ na ukazatele na Objekt Descriptor
0x14	dwUnused2	nevyužité
0x18	szProjectDescription	ukazateľ na popis projektu
0x1C	szProjectHelpFile	ukazateľ na help súbor
0x20	dwReserved2	nevyužité, vždy -1
0x24	dwHelpContextId	Help Context ID nastavené v nastaveniach projektu

Tabuľka 8.6 Sekundárne informácie o projekte (The Secondary Project Information)

```
Project info 2:  
lpHeapLink          0  
lpObjectTable      401274  
dwReserved         -1  
dwUnused           0  
lpObjectList       401920  
dwUnused2          0  
szProjectDescription 0  
szProjectHelpFile  0  
dwReserved2       -1  
dwHelpContextId    0
```

8.3.4 Tabuľka objektov

Štruktúra Tabuľka objektov obsahuje ukazateľ na Pole objektov ako aj opakované Dáta o projekte, pravdepodobne pre rýchlejší prístup k nim. Niektoré z týchto dát sa využívajú len keď je projekt spustený v pamäti (v IDE).

Na štruktúru Tabuľka objektov ukazuje ukazateľ lpObjectTable, ktorý sa nachádza v štruktúre Informácie o projekte.

Tabuľka objektov je štruktúra o veľkosti 0x54 = 84 bajtov. V tabuľke 8.7 je znázomená jej štruktúra.

Offset	Názov	Popis
0x0	lpHeapLink	nevyužitý, vždy 0
0x4	lpExecProj	ukazateľ na VB Project Exec COM Object
0x8	lpProjectInfo2	ukazateľ na Sekundárne Informácie o Projekte
0xC	dwReserved	nevyužitý, vždy -1
0x10	dwNull	nevyužitý
0x14	lpProjectObject	ukazateľ na Project Data v pamäti
0x18	uuidObject	GUID Objektovej Tabuľky
0x28	fCompileState	interný príznak používaný počas kompilácie
0x2A	dwTotalObjects	počet objektov v projekte
0x2C	dwCompiledObjects	po kompilovaní rovnaké ako dwTotalObjects
0x2E	dwObjectsInUse	po kompilovaní obyčajne rovnaké ako dwTotalObjects
0x30	lpObjectArray	ukazateľ na Public Object Descriptors
0x34	fIdeFlag	príznak použitý v IDE
0x38	lpIdeData	príznak použitý v IDE
0x3C	lpIdeData2	príznak použitý v IDE
0x40	lpszProjectName	ukazateľ na meno projektu
0x44	dwLcid	LCID projektu
0x48	dwLcid2	alternatívne LCID projektu
0x4C	lpIdeData3	príznak použitý v IDE
0x50	dwIdentifier	verzia predlohy štruktúry

Tabuľka 8.7 Tabuľka objektov (The Object Table)

Object table:

lpHeapLink	0
lpExecProj	402024
lpProjectInfo2	40196C
dwReserved	-1
dwNull2	0
lpProjectObject	402014
uuidObject	D6-A1-3E-B7-CA-55-1D-4E-83-6B-E4-21-9A-60-1C-B2
fCompileState	A
dwTotalObjects	1
dwCompiledObjects	1
dwObjectsInUse	1
lpObjectArray	4012C8
fIdeFlag	0
lpIdeData	0
lpIdeData2	0

```

lpzProjectName  401308
dwLcid          409
dwLcid2         409
lpIdeData3     0
dwIdentifier    2

```

8.3.5 Verejný deskriptor objektu

Štruktúra Verejný deskriptor objektu obsahuje ukazateľ na Pole objektov a opakovane aj Dáta o projekte, pravdepodobne pre rýchlejší prístup k nim. Niektoré z týchto dát sa využívajú len keď je projekt spustený v pamäti (v IDE).

Na štruktúru Verejný deskriptor objektu ukazuje ukazateľ lpObjectArray, ktorý sa nachádza v štruktúre Tabuľka objektov.

Verejný deskriptor objektu je štruktúra o veľkosti 0x54 = 84 bajtov. V tabuľke 8.8 je znázornená jej štruktúra.

Offset	Názov	Popis
0x0	lpObjectInfo	ukazateľ na informácie o objekte pre tento objekt
0x4	dwReserved	po kompilácii vždy -1
0x8	lpPublicBytes	ukazateľ na Public Integers premennej veľkosti
0xC	lpStaticBytes	ukazateľ na Static Integers premennej veľkosti
0x10	lpModulePublic	ukazateľ na Public premenné v sekcii DATA
0x14	lpModuleStatic	ukazateľ na Static premenné v sekcii DATA
0x18	lpzObjectName	názov objektu
0x1C	dwMethodCount	počet metód v objekte
0x20	lpMethodNames	ukazateľ na pole názvov metód
0x24	bStaticVars	offset, kde kopírovať Static premenné
0x28	fObjectType	príznaky definujúce typ objektu
0x2C	dwNull	nevyužitá

Tabuľka 8.8 Verejný deskriptor objektu (The Public Object Descriptor)

Public Object Descriptor:

```

lpObjectInfo      401698
dwReserved3       -1
lpPublicBytes     401408
lpStaticBytes     0
lpModulePublic    0
lpModuleStatic    0
lpzObjectName     401300
dwMethodCount     2
lpMethodNames     4012F8

```

```

bStaticVars          FFFF
fObjectType           18083
dwNull13             0

```

8.3.6 Informácie o objekte

Štruktúra Informácie o objekte definuje objekty a obsahuje rozličné informácie o jeho metódach a konštantách (v P-kóde).

Na štruktúru Informácie o objekte ukazuje ukazateľ `lpObjectInfo`, ktorý sa nachádza v štruktúre Verejný deskriptor objektu.

Informácie o objekte je štruktúra o veľkosti $0x38 = 56$ bajtov. V tabuľke 8.9 je znázornená jej štruktúra.

Ofset	Názov	Popis
0x0	wRefCount	po kompilácii vždy 1
0x2	wObjectIndex	index tohto objektu
0x4	lpObjectTable	ukazateľ na Tabuľku objektov
0x8	lpIdeData	po kompilácii 0, využité len v IDE
0xC	lpPrivateObject	ukazateľ na Privátny deskriptor objektu
0x10	dwReserved	po kompilácii vždy -1
0x14	dwNull	nevyužité
0x18	lpObject	ukazateľ na Verejný deskriptor objektu
0x1C	lpProjectData	ukazateľ na Projektový objekt v pamäti
0x20	wMethodCount	počet metód
0x22	wMethodCount2	po kompilácii 0, využité len v IDE
0x24	lpMethods	ukazateľ na pole metód
0x28	wConstants	počet konštant bloku konštant
0x2A	wMaxConstants	počet konštant na alokovanie v bloku konštant
0x2C	lpIdeData2	využité len v IDE
0x30	lpIdeData3	využité len v IDE
0x34	lpConstants	ukazateľ na blok konštant

Tabuľka 8.9 Informácie o objekte (The Object Info)

Object info:

```

wRefCount          1
wObjectIndex      0
lpObjectTable     401274
lpIdeData         0
lpPrivateObject   40192C
dwReserved        -1
dwNull            0

```


Ofset	Názov	Popis
0x0	lpHeapLink	nevyužitý, po kompilácii vždy 0
0x4	lpObjectInfo	ukazateľ na informácie o objekte pre tento objekt
0x8	dwReserved	po kompilácii vždy -1
0xC	dwIdeData[3]	po kompilácii nevyužitý
0x18	lpObjectList	ukazateľ na nadradenú štruktúru (pole)
0x1C	dwIdeData2	po kompilácii nevyužitý
0x20	lpObjectList2[3]	ukazateľ na nadradenú štruktúru (pole)
0x2C	dwIdeData3[3]	po kompilácii nevyužitý
0x38	dwObjectType	typ opisovaného objektu
0x3C	dwIdentifier	verzia predlohy štruktúry

Tabuľka 8.10 Privátny deskriptor objektu (The Private Object Descriptor)

8.3.8 COM Registračné dáta

COM Registračné dáta obsahujú informácie o tom, či daný súbor je ActiveX. Ďalej obsahuje dôležité COM Registračné dáta ako Typelib informácie, Designer dáta a CLSID rozhraní.

Na štruktúru COM Registračné dáta ukazuje ukazateľ lpComRegisterData, ktorý sa nachádza v štruktúre Visual Basic hlavička.

COM Registračné dáta je štruktúra o veľkosti 0x2A = 42 bajtov. V tabuľke 8.11 je znázornená jej štruktúra.

Ofset	Názov	Popis
0x0	bRegInfo	ofset na COM interfejs info
0x4	bSZProjectName	ofset na názov projektu
0x8	bSZHelpDirectory	ofset na Help adresár
0xC	bSZProjectDescription	ofset na popis projektu
0x10	uuidProjectClsId	CLSID projektu
0x20	dwTlbLcid	LCID typu knižnice
0x24	wUnknown	neznáme
0x26	wTlbVerMajor	verzia Typelib
0x28	wTlbVerMinor	subverzia Typelib

Tabuľka 8.11 COM Registračné dáta (The COM Registration Data)

COM Register Data:

```

bRegInfo           0
bSZProjectName     30
bSZHelpDirectory   40
bSZProjectDescription  0

```

```

uuidProjectClsId      A5-B3-29-48-E2-72-75-49-A2-B9-28-74-1B-56-B6-48
dwTlbLcid             0
wUnknown              0
wTlbVerMajor         1
wTlbVerMinor         0

ProjectName           Project1
HelpDirectory

```

8.3.9 COM Registračné informácie

Ak je objekt platný a potrebuje byť registrovaný tak štruktúra COM Registračné informácie obsahuje informácie o tomto objekte.

Na štruktúru COM Registračné informácie ukazuje ukazateľ `bRegInfo`, ktorý sa nachádza v štruktúre COM Registračné dáta. Ak je tento ukazateľ rovný 0, potom neexistuje žiadna štruktúra COM Registračné informácie. V opačnom prípade hodnota `bRegInfo` je ofset od začiatku štruktúry COM Registračné dáta.

COM Registračné informácie je štruktúra o veľkosti $0x44 = 68$ bajtov. V tabuľke 8.12 je znázomená jej štruktúra.

Ofset	Názov	Popis
0x0	<code>bNextObject</code>	ofset na ďalšie COM interfejs info
0x4	<code>bObjectName</code>	ofset na názov objektu
0x8	<code>bObjectDescription</code>	ofset na popis objektu
0xC	<code>dwInstancing</code>	Instancing mód
0x10	<code>dwObjectId</code>	aktuálne ID objektu v projekte
0x14	<code>uuidObject</code>	CLSID objektu
0x24	<code>fIsInterface</code>	špecifikuje, či ďalšie CLSID je validne
0x28	<code>bUuidObjectIFace</code>	ofset na CLSID objektového interfejsu
0x2C	<code>bUuidEventsIFace</code>	ofset na CLSID udalostného interfejsu
0x30	<code>fHasEvents</code>	špecifikuje, či predchádzajúce CLSID je validne
0x34	<code>dwMiscStatus</code>	OLEMISC príznaky
0x38	<code>fClassType</code>	typ triedy
0x39	<code>fObjectType</code>	príznaky identifikujúce typ objektu
0x3A	<code>wToolboxBitmap32</code>	ID kontrólnej bitmapy v Toolboxe
0x3C	<code>wDefaultIcon</code>	ikona pri minimalizovanom okne Control
0x3E	<code>fIsDesigner</code>	špecifikuje, či sa jedná o Designer
0x40	<code>bDesignerData</code>	ofset na Designer dáta

Tabuľka 8.12 COM Registračné informácie (The COM Registration Info)

There is no COM Registration Info present in this file.

Ofset	Názov	Popis
0x2	Designer	Visual Basic Designer pre Add-In
0x10	Class Module	Trieda Visual Basic
0x20	User Control	Visual Basic Active X User Control (OCX)
0x80	User Document	Visual Basic užívateľský dokument

Tabuľka 8.13 Príznyky identifikujúce typ objektu vo fObjectType

Môžu existovať aj iné príznaky identifikujúce Visual Basic Objekty, avšak tieto sa v danej štruktúre nevyskytujú.

8.3.10 COM Designer informácie

Ak je objekt typu Designer (používané pre Add-In) potom je štruktúra COM Registračné informácie nasledovaná štruktúrou COM Designer.

COM Designer informácie je štruktúra premenlivej veľkosti. V tabuľke 8.14 je znázornená jej štruktúra.

Ofset	Názov	Popis
0x0	uuidDesigner	Addin/Designer CLSID
0x10	cbStructSize	celková veľkosť nasledujúcich polí
0x14	bstrAddinRegKey	registračný kľúč Addin
VAR	bstrAddinName	názov Addin
VAR	bstrAddinDescription	popis Addin
VAR	dwLoadBehaviour	CLSID objektu
VAR	bstrSatelliteDll	satelitné DLL, ak je špecifikované
VAR	bstrAdditionalRegKey	Extra registry kľúč, ak je špecifikovaný
VAR	dwCommandLineSafe	ak je 1, špecifikuje Addin bez GUI

Tabuľka 8.14 COM Designer informácie (The COM Designer Info)

Pri čítaní štruktúry COM Designer informácie jednoducho načítame hodnotu na ofsete 0x14, čo je dĺžka Add-In registračného kľúča. Túto hodnotu potom pripočítame ku súčasnému ofsetu a tak dostaneme ofset dĺžky názvu Add-In. Ak toto číslo znova pripočítame k nášmu novému ofsetu, dostaneme sa na popis Add-In. Nasledujúca hodnota je Load Behavior, dĺžky 4 bajty nasledovaná dĺžkou názvu setelitného DLL. Ak je táto hodnota rovná 0, tak sa jedná o Extra registry kľúč. Ak je aj táto hodnota rovná 0, jedná sa o hodnotu dwCommandLineSafe. [17][26][27][29][30][31]

8.3.11 Prehľad štruktúry binárneho súboru Visual Basic 6

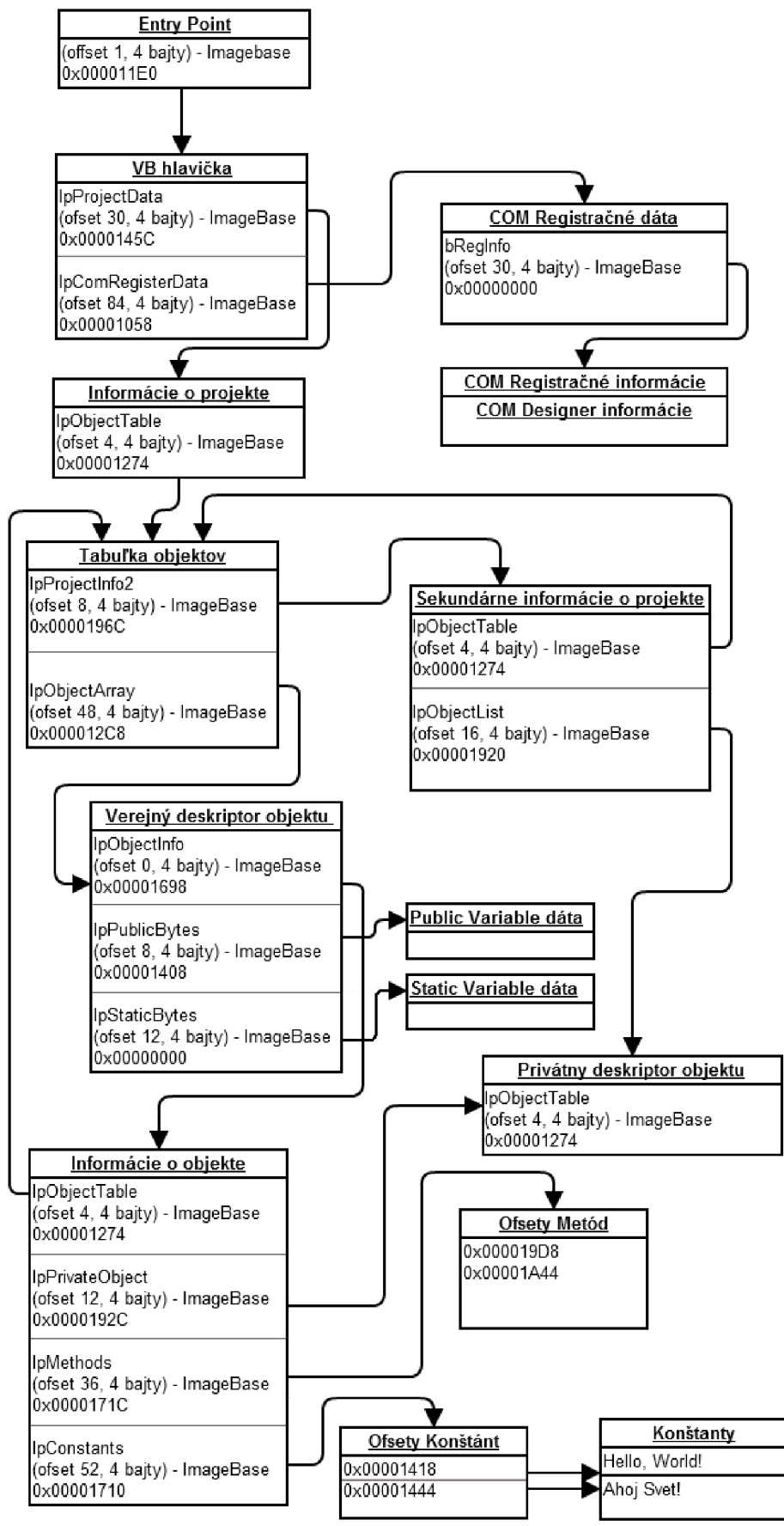
Pomocou predchádzajúcich informácií sme schopní načrtnúť štruktúru binárneho súboru Visual Basic 6, ktorú môžeme vidieť na obrázku 8.18.

Tento obrázok detailne popisuje prepojenie štruktúr Visual Basic 6. Sú v ňom uvedené konkrétne hodnoty ofsetov, na ktorých sa nachádzajú informácie o ďalších štruktúrach. Taktiež sú tu uvedené konkrétne hodnoty z referenčného súboru skompilovaného pod Visual Basic 6 „p-code2.exe“.

Informácie sú uvedené vo formáte uvedenom na obrázku 8.17.

Názov štruktúry
Názov položky v štruktúre (Ofset položky v štruktúre, dĺžka položky) – ak od tejto hodnoty odrátame ImageBase dostaneme adresu štruktúry, na ktorú ukazuje šípka v obrázku Konkrétne adresa odkazovanej štruktúry v súbore „p-code2.exe“

Obr. 8.17 Formát uvedených informácií na obrázku 8.18



Obr. 8.18 Štruktúra binárneho súboru Visual Basic 6

8.4 Praktické riešenie

Program je implementovaný v jazyku C# vo vývojovom prostredí Microsoft Visual Studio 2010 pod operačným systémom Microsoft Windows 7 64bit.

Pri jeho vývoji sa vyskytli mnohé problémy. V dobe vývoja tohto programu neexistoval žiaden prehľad o štruktúre binárnych súborov Visual Basic 6 a preto bol celý vývoj tohto programu spojený s poznávaním a analýzou štruktúr binárnych súborov Visual Basic 6.

Okrem tohto problému sa vyskytli mnohé menšie problémy ako bol napríklad problém načítavania UNICODE textového reťazca bez poznania jeho dĺžky. UNICODE textový reťazec je v binárnom súbore definovaný 2 bajtmi.

00001486	2A 00 5C 00 41 00 63 00	3A 00 5C 00 50 00 72 00	*.\.A.c.:\.P.r.
00001490	6F 00 67 00 72 00 61 00	6D 00 20 00 46 00 69 00	o.g.r.a.m. .F.i.
000014A0	6C 00 65 00 73 00 5C 00	4D 00 69 00 63 00 72 00	l.e.s.\.M.i.c.r.
000014B0	6F 00 73 00 6F 00 66 00	74 00 20 00 56 00 69 00	o.s.o.f.t. .V.i.
000014C0	73 00 75 00 61 00 6C 00	20 00 53 00 74 00 75 00	s.u.a.l. .S.t.u.
000014D0	64 00 69 00 6F 00 5C 00	56 00 42 00 39 00 38 00	d.i.o.\.V.B.9.8.
000014E0	5C 00 50 00 72 00 6F 00	6A 00 65 00 63 00 74 00	\.P.r.o.j.e.c.t.
000014F9	31 00 2E 00 76 00 62 00	70 00 00 00 00 00 00 00	1...v.b.p.....

Obr. 8.19 Unicode textový reťazec v binárnom súbore

Okno programu sa skladá z dvoch záložiek, sú to záložka Main a záložka Details. V záložke Main po stlačení tlačidla browse sa nám otvorí dialógové okno pre výber súboru na analýzu. Po stlačení tlačidla disassemble program zanalyzuje všetky známe štruktúry binárnych súborov Visual Basic 6 a najdôležitejšie informácie ako aj hlavičku Visual Basic vypíše do okna Main info. V okne Imports sú vypísané dll knižnice a funkcie, ktoré program využíva pri svojom behu. Ukážku môžeme vidieť na obrázku 8.20.

8.5 Praktické využitie

Pomocou tohto programu vieme získať textové reťazce, konštanty a ďalšie užitočné informácie používané v programoch kompilovaných pomocou Visual Basic 6.

Existujú funkčné algoritmické detekcie, ktoré na základe týchto informácií dokážu vyhodnotiť, či sa jedná o štandardný neškodný program alebo o užívateľovi škodlivý program malware. Toto je možné vyhodnocovať, na základe používania rovnakých prvkov, ktoré sú pri mnohých škodlivých programoch nutnosťou alebo aj lenivosťou programátorov malware. V mnohých prípadoch sa jedná o vývoj programov zo šablón, kde je možné s minimálnym úsilím program prispôbiť svojim požiadavkám na funkčnosť. Poznanie štruktúry týchto šablón výrazne napomáha detekcií takto vytvorených súborov malware. Jedná sa napríklad o typické škodlivé súbory typu banker, ktoré sú veľmi rozšírené v Brazílii.

Taktiež do tejto kategórie spadajú aj škodlivé súbory typu „fakeanti“. Jedná sa o programy, ktoré vyzerajú ako známe Antivírusy alebo Anti Spam programy, avšak nimi zobrazené výsledky slúžia na zmätenie užívateľa. Tieto programy zobrazujú výstrahy napadnutia počítača, ktorý v skutočnosti napadnutý nie je. Po zakúpení licencie sľubujú odstránenie týchto napadnutí. Užívateľ v mnohých prípadoch týmto výstrahám uverí a daný „fakeanti“ program si zakúpi v domnení, že tak svojmu počítaču pomôže. Opak je však pravdou.

9 Záver

Cieľom tejto diplomovej práce bolo zanalyzovať binárny formát programovacieho jazyka Visual Basic, zorientovať sa v kódach do akých vie kompilovať, navrhnúť spôsob analýzy a disasemblingu týchto súborov. Najväčším problémom bola nedostupnosť akýchkoľvek verejných dokumentov, ktoré by sa zaoberali štruktúrou týchto súborov. Firma Microsoft, ktorá tento formát navrhla a vyvíjala nezostavila žiadnu praktickú dokumentáciu. Nakoľko je tento formát zastaraný a v dnešnej dobe už pri bežnom programovaní takmer nepoužívaný, neexistuje takmer žiaden záujem o jeho spätnú špecifikáciu. Aj napriek tomu sa tento formát podarilo zanalyzovať a bol navrhnutý spôsob ďalšej analýzy binárnych súborov Visual Basic 6 pomocou existujúcich nástrojov. Na základe tejto analýzy bol podrobne popísaný binárny formát Visual Basic 6. Táto analýza obsahovala zhrnutie množstva poznatkov nadobudnutých čítaním mnohých fór, využívaním metód reverzného inžinierstva pri analýze samotných Visual Basic 6 súborov, ako aj analýze iných programov na analýzu týchto súborov. Rovnako si táto analýza vyžadovala neustále skúmanie viacerých binárnych súborov Visual Basic 6 a premýšľanie o tom, čo môžu jednotlivé položky, ofsety a adresy označovať.

Prínosom tejto práce je podrobný popis štruktúr, ktoré sa nachádzajú v binárnych súboroch skompilovaných pomocou Visual Basic 6. Bol vytvorený podrobný prehľad týchto štruktúr a taktiež popísaný presný postup ako sa ku konkrétnym informáciám, položkám a premenným dopracovať. Rovnako sú uvedené informácie, kde v spustiteľnom súbore sa tieto položky nachádzajú a ako ich spracovať.

Pomocou týchto informácií bol implementovaný nástroj, ktorý dokáže tieto štruktúry spracovať a získať dáta veľmi potrebné na analýzu binárnych súborov. Pomocou týchto dát na základe rôznych heuristických metód je možné s veľkou pravdepodobnosťou určiť, či sa jedná o bežný Visual Basic program alebo či sa jedná o užívateľovi škodlivý súbor malware. Funkčnosť tohto nástroja bola overená na rôznych súboroch skompilovaných v prostredí Visual Basic 6 ako aj rôznych užívateľovi škodlivých súborov malware.

Prínosom pre autora bolo zoznámenie sa so štruktúrou spustiteľných súborov, s ich analýzou a spracovaním štruktúr a informácií, ktoré tieto súbory obsahujú. Rovnako bolo prínosom aj pochopenie princípov spúšťania spustiteľných súborov pod operačným systémom Windows a princípy fungovania binárnych súborov Visual Basic 6 a mnoho iných informácií o spustiteľných súboroch, vrátane binárnych súborov VB 6.

V prípade pokračovania práce na tomto nástroji je možné skúmať štruktúry binárnych súborov Visual Basic do väčšej hĺbky a tým získať viac informácií o súboroch, na základe ktorých by bolo možné presnejšie určiť, čo konkrétny program robí bez toho aby bol spustený. Pomocou informácií uvedených v práci je možné pokračovať v ďalšej podrobnejšej analýze a získať informácie o ďalších štruktúrach, procedúrach, premenných a funkciách, ktoré sa v binárnom súbore Visual Basic 6 ešte môžu vyskytovať. Rovnako je možné pochopiť prepojenie a nadväznosť týchto štruktúr a vytvoriť komplexný pohľad na funkcionality analyzovaných súborov.

Vytvorený nástroj pomôže pri statickej analýze binárnych súborov Visual Basic 6 a znalosť štruktúr týchto binárnych súborov napomôže ďalšiemu vývoju nástrojov na automatickú heuristickú analýzu.

Literatúra

- [1] Microsoft Portable Executable and Common Object File Format Specification [online]. 2010-09-21 [cit. 2011-01-01] Dostupné na URL: <http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.msp>
- [2] The Portable Executable File Format [online]. [cit. 1.1.2011] Dostupný z WWW: <http://www.csn.ul.ie/~caolan/publink/winresdump/winresdump/doc/pefile.html>
- [3] Tiny PE, Creating the smallest possible PE executable [online]. [cit. 1.1.2011] Dostupný z WWW: <http://www.phreedom.org/solar/code/tinype/>
- [4] McKinney, Bruce, 1953: *Hardcore Visual Basic 2nd ed.* Redmond, Washington, Microsoft Press, 1997. ISBN 1-57231-422-2
- [5] Eilam, Eldad: *Reversing: Secrets of Reverse Engineering.* Indianapolis, Wiley Publishing, Inc., 2005. ISBN-10: 0-7645-7481-7, ISBN-13: 978-0-7645-7481-8
- [6] Vrátil, Zdeněk: *Assembler PC.* Sokolov, GETHON audio and computer, 1997, 442 s.
- [7] Eagle, Chris: *The IDA Pro Book.* San Francisco, No Starch Press, Inc., 2008. ISBN-10: 1-59327-178-6, ISBN-13: 978-1-59327-178-7
- [8] Schwarz, Benjamin; Debray, Saumya; Andrews, Gregory: *Disassembly of Executable Code Revisited.* 10 s.
- [9] Sulaiman, A.; Ramamoorthy, K.; Mukkamala, S.; Sung, A.H.: *Disassembled Code Analyzer for Malware.* 6 s.
- [10] Petzold, Charles: *Programming Windows Fifth Edition.* Redmond, Washington, Microsoft Press, 1998. ISBN 1-57231-995-X
- [11] Overview of the Windows API [online]. [cit. 2011-01-07] Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/Aa383723>
- [12] Yason, Mark Vincent: The Art of unpacking. [online]. [cit. 2011-01-07] Dostupný z WWW: <http://www.blackhat.com/presentations/bh-usa-07/Yason/Whitepaper/bh-usa-07-yason-WP.pdf>
- [13] Atkinson, Ian: The Microsoft Windows API, 2000. [online]. [cit. 2011-01-07] Dostupný z WWW: http://myweb.msoc.edu/~barnicks/courses/cs384/papers19992000/atkinsoi-atkinsoi_term_paper.pdf
- [14] Visual Basic Concepts, Compiling Your Project to Native Code [online]. [cit. 2011-01-08] Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/aa240843.aspx>
- [15] Eriksson, David: Designing an object-oriented decompiler: Decompilation support for Interactive Disassembler Pro [online]. [cit. 2011-01-08] Dostupný z WWW: http://desquirr.sourceforge.net/desquirr/desquirr_master_thesis.pdf
- [16] P-Code Versus Native Code [online]. [cit. 2011-01-08] Dostupný z WWW: <http://vb.mvps.org/hardcore/html/p-codeversusnativecode.htm>
- [17] Karve, Sanchit: DISASSEMBLING VISUAL BASIC APPLICATIONS [online]. [cit. 2011-01-08] Dostupný z WWW: http://www.dreamincode.net/forums/index.php?app=core&module=attach§ion=attach&attach_id=3198&s=4d1b6e454afa142c3d9156729e9fb751

- [18] Watanabe, Tsuyoshi: How to write a disassembler [online]. 1999-01-XX [cit. 2011-01-09] Dostupný z WWW: <http://www.spiralspace.com/Depot/Projects/Disassembler/Default.aspx>
- [19] Iczelion's Win32 Assembly Tutorial 6: Import Table [online]. [cit. 2011-04-05] Dostupný z WWW: <http://win32assembly.online.fr/pe-tut6.html>
- [20] PE Files Import Table Rebuilding [online]. 1999-07-18 [cit. 2011-04-25] Dostupný z WWW: http://www.reverse-engineering.info/PE_Information/rebuild.txt
- [21] THE PORTABLE EXECUTABLE FORMAT [online]. [cit. 2011-04-27] Dostupný z WWW: <http://www.nikse.dk/petxt.html>
- [22] Albertini, Ange: the PE format [online]. [cit. 2011-04-29] Dostupný z WWW: <http://corkami.googlecode.com/files/pe.pdf>
- [23] PE File Structure [online]. [cit. 2011-05-02] Dostupný z WWW: <http://www.thehackademy.net/madchat/vxdevl/papers/winsys/pefile/pefile.htm>
- [24] Danehkar, Ashkbiz: Inject Your Code to a Portable Executable File [online]. [cit. 2011-05-04] Dostupný z WWW: <http://www.codeguru.com/cpp/w-p/system/misc/article.php/c11393>
- [25] MSDN IMAGE_DATA_DIRECTORY Structure [online]. [cit. 2011-05-06] Dostupný z WWW: [http://msdn.microsoft.com/en-us/library/ms680305\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms680305(v=vs.85).aspx)
- [26] Ionescu, Alex: Visual Basic Image Internal Structure Format [online]. [cit. 2011-05-15] Dostupný z WWW: <http://www.alex-ionescu.com/vb.pdf>
- [27] Decompiler-VB.net - VBReFormer USER MANUAL [online]. [cit. 2011-05-15] Dostupný z WWW: <http://www.decompiler-vb.net/documentation%5CVBReFormer%20-%20Help.pdf>
- [28] Scambray, Joel; McClure, Stuart; Kurtz, George: *Hacking bez tajemství. 2. Aktualizované vydání*. Praha Computer Press, 2000. Xxv, 625 s. ISBN-80-7226-644-6
- [29] Nikishin, Andy; Pavlyushchik, Mike: Generic Detection for Visual Basic Internet Worms [online]. [cit. 2011-05-16] Dostupný z WWW: <http://www.virusbtn.com/pdf/magazine/2002/200201.pdf>
- [30] Marko, Richard, ESET Software, Slovakia: VB Wearing the Inside Out [online]. [cit. 2011-05-16] Dostupný z WWW: <http://www.virusbtn.com/pdf/magazine/2002/200206.pdf>
- [31] Wong, Reginald: vb.idc (for Visual Basic 5/6) [online]. [cit. 2011-05-16] Dostupný z WWW: <http://www.hex-rays.com/idapro/freefiles/vb.idc>
- [32] Choi, Yang-seo; Kim, Ik-kyun; Oh, Jin-tae; Ryou Jae-cheol: PE File Header Analysis-Based Packed PE File Detection Technique (PHAD) [online]. [cit. 2011-05-16] Dostupný z WWW: <http://www.computer.org/portal/web/csdl/doi/10.1109/CSA.2008.28>