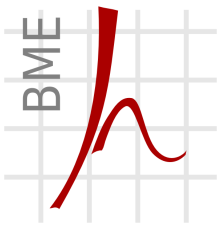
Dinamikus linkelés & IDA & Program Instrumetation

Kódvisszafejtés.



Híradástechnikai Tanszék

Izsó Tamás

2016. december 7.

SECTION TABLE

A section táblák kezdete:

```
pSectionTable = &NtHeader.OptionalHeader +
                NtHeader.FileHeader.SizeOfOptionalHeader;
```

```
typedef struct _IMAGE_SECTION_HEADER {
    BYTE    Name[IMAGE_SIZEOF_SHORT_NAME]; // section neve
    union {
            DWORD   PhysicalAddress;
            DWORD   VirtualSize;           // lefoglalt méret
    } Misc;
    DWORD   VirtualAddress;                // RVA
    DWORD   SizeOfRawData;                 // adatok tényleges mérete
    DWORD   PointerToRawData;
    DWORD   PointerToRelocations;
    DWORD   PointerToLinenumbers;
    WORD    NumberOfRelocations;
    WORD    NumberOfLinenumbers;
    DWORD   Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

SECTION TABLE dumpja

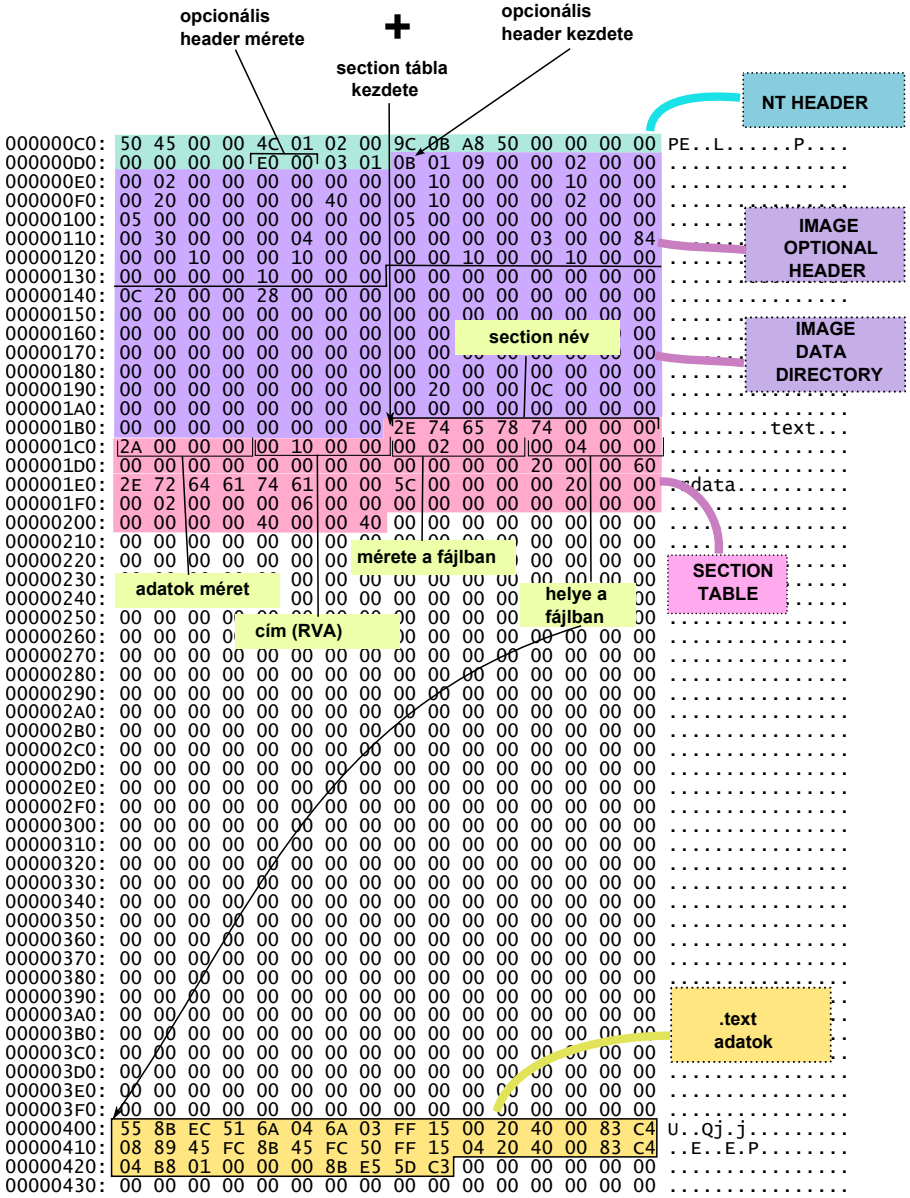


IMAGE DATA DIRECTORY – 16 db bejegyzés

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

IMAGE DATA Bejegyzések – 16 db bejegyzés

0	Export Table	dll .edata Section
1	Import Table	dll .idata
2	Resource Table	.rsrc Section
3	Exception Table	.pdata Section
4	Certificate Table	
5	Base Relocation Table	.reloc Section
6	Debug	.debug Section
7	Architecture Specific Data	reserved 0
8	Global Ptr	
9	TLS Directory	
10	Load Configuration Directory	
11	Bound Import	
12	IAT	import address table
13	Delay Load Import Descriptors	
14	CLR Runtime descriptor	
15	reserved	0

IMAGE DATA kikeresése a SECTION TABLE-ban

A section tábla kezdete a memóriában (RVA):

```
PIMAGE_SECTION_HEADER getSectionHdr( DWORD rva )
{
    PIMAGE_SECTION_HEADER section = IMAGE_FIRST_SECTION(pNTHeader);
    unsigned i;

    for(i=0; i < pNTHeader->FileHeader.NumberOfSections; i++, section++) {
        DWORD size = section->Misc.VirtualSize;

        // Is the RVA within this section?
        if( (rva >= section->VirtualAddress) &&
            (rva < (section->VirtualAddress + size)))
          return section;
    }
    return 0;
}
```

A section tábla kezdete a fájlban (offset):

```
DWORD GetOffsetFromRVA( DWORD rva, PIMAGE_SECTION_HEADER pSectionHdr)
{
    // rva = 0x200C
    // Dumpban 0x2000 ↦ 0x600, delta = 0x0C
    DWORD delta = rva - pSectionHdr->VirtualAddress;
    return  delta + pSectionHdr->PointerToRawData; // 0x60C
}
```

EXPORT DIRECTORY Entry

```
typedef struct _IMAGE_EXPORT_DIRECTORY {
    DWORD   Characteristics;          // Set 0
    DWORD   TimeDateStamp;
    WORD    MajorVersion;             // User set
    WORD    MinorVersion;             // User set
    DWORD   Name;                     // DLL name (RVA)
    DWORD   Base;
    DWORD   NumberOfFunctions;
    DWORD   NumberOfNames;
    DWORD   AddressOfFunctions;       // EAT (RVA)
    DWORD   AddressOfNames;           // RVA
    DWORD   AddressOfNameOrdinals;    // RVA
} IMAGE_EXPORT_DIRECTORY, *PIMAGE_EXPORT_DIRECTORY;
```

IMPORT DIRECTORY DATA dumpja

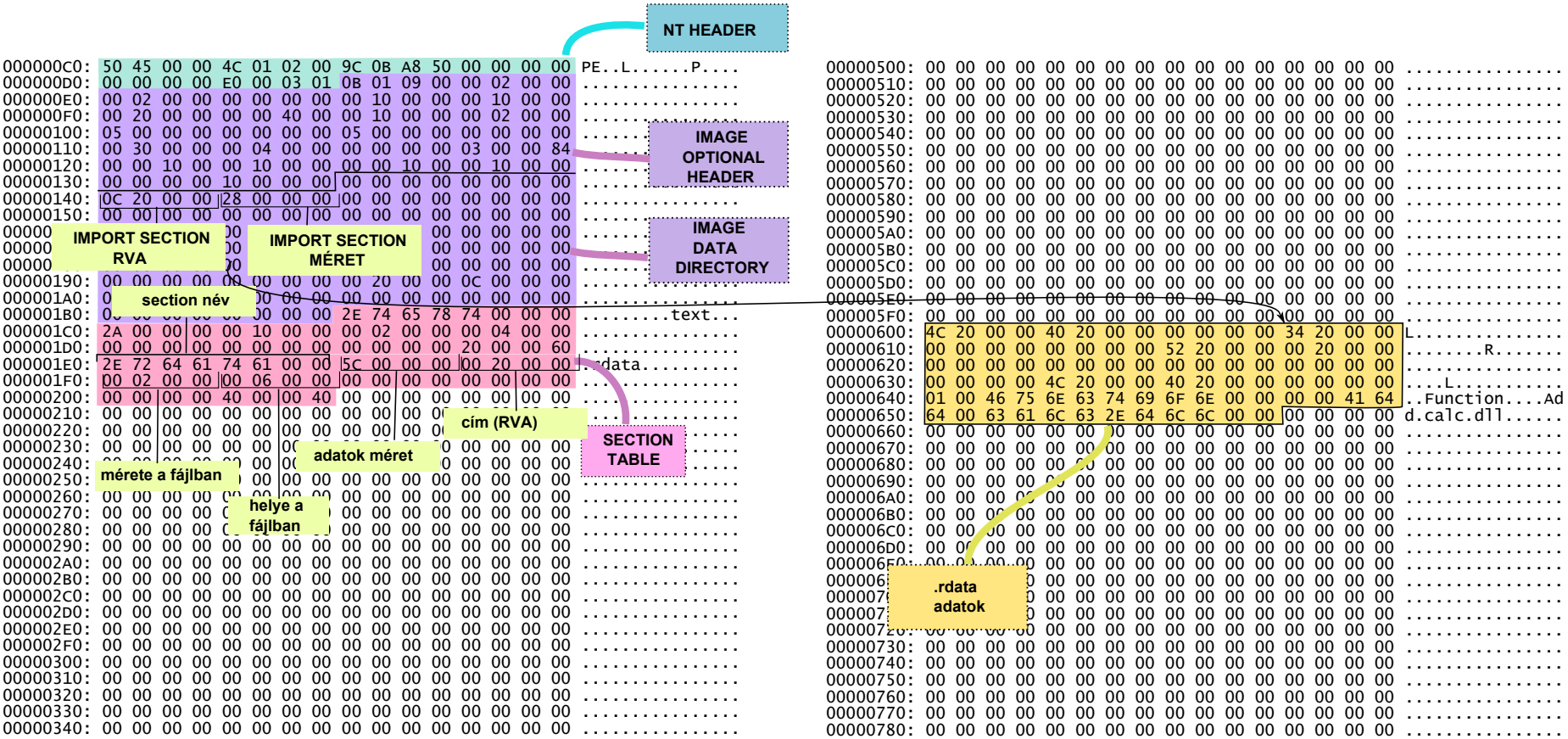
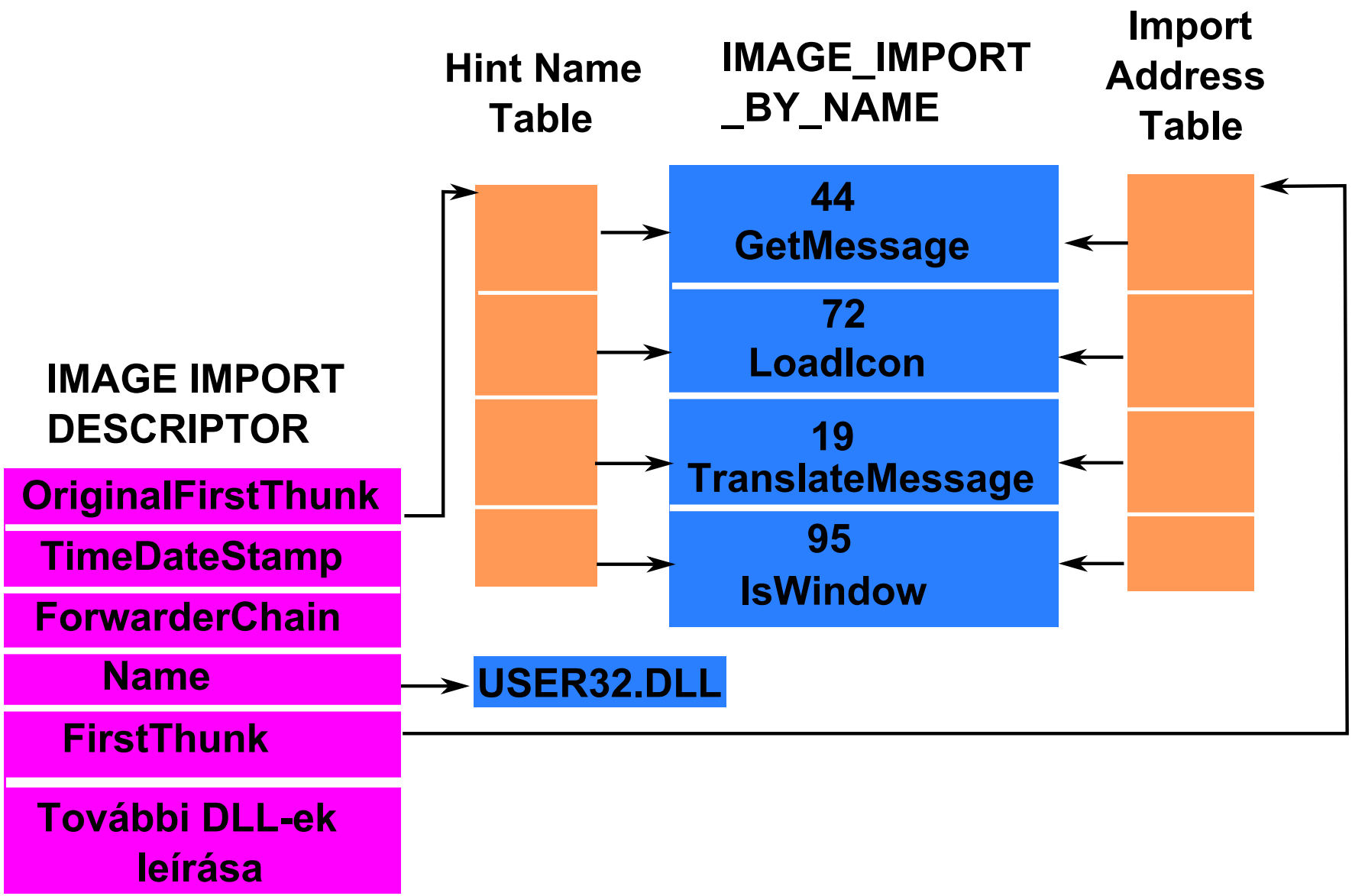


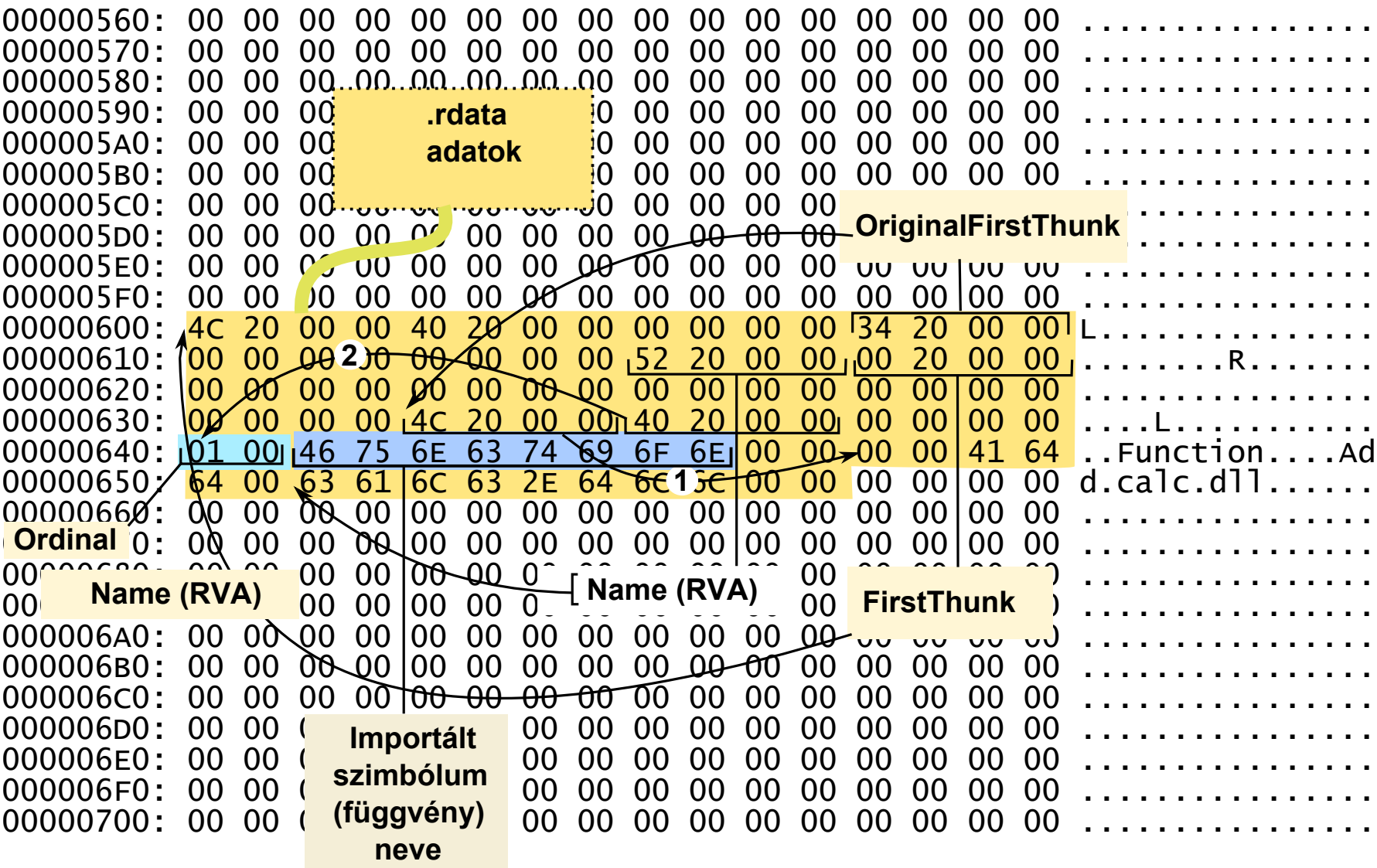
IMAGE IMPORT DESCRIPTOR

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD    Characteristics;
        DWORD    OriginalFirstThunk; // RVA to original unbound IAT
    } DUMMYUNIONNAME;
    DWORD    TimeDateStamp;          // 0 if not bound,

    DWORD    ForwarderChain;         // -1 if no forwarders
    DWORD    Name;
    DWORD    FirstThunk;             // RVA to IAT
} IMAGE_IMPORT_DESCRIPTOR;

typedef struct _IMAGE_IMPORT_BY_NAME {
     WORD       Hint;
     BYTE       Name[1];
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

IMPORT TABLE



IMPORT TABLE



IMPORT TABLE – program indítása után a memóriában

```
00402000  00 10 00 10|30 10 00 10|00 00 00 00|34 20 00 00|  ....0.......4 ..
00402010  00 00 00 00|00 00 00 00|52 20 00 00|00 20 00 00|  ........R ... ..
00402020  00 00 00 00|00 00 00 00|00 00 00 00|00 00 00 00|  ................
00402030  00 00 00 00|4C 20 00 00|40 20 00 00|00 00 00 00|  ....L ... ......
00402040  01 00 46 75|6E 63 74 69|6F 6E 00 00|00 00 41 64|  ..Function....Ad
00402050  64 00 63 61|6C 63 2E 64|6C 6C 00 00|00 00 00 00|  d.calc.dll......
```

DLL-ben lévő függvény hívása

```
00401000    55              PUSH EBP
00401001    8BEC            MOV EBP,ESP
00401003    51              PUSH ECX
00401004    6A 04           PUSH 4
00401006    6A 03           PUSH 3
00401008    FF15 00204000   CALL DWORD PTR DS:[<&calc.Add>]
0040100E    83C4 08         ADD ESP,8
00401011    8945 FC         MOV DWORD PTR SS:[LOCAL.1],EAX
00401014    8B45 FC         MOV EAX,DWORD PTR SS:[LOCAL.1]
00401017    50              PUSH EAX
00401018    FF15 04204000   CALL DWORD PTR DS:[<&calc.Function>]
0040101E    83C4 04         ADD ESP,4
00401021    B8 01000000     MOV EAX,1
00401026    8BE5            MOV ESP,EBP
00401028    5D              POP EBP
00401029    C3              RETN
```

DLL tartalma

```
10001000    55              PUSH EBP
10001001    8BEC            MOV EBP,ESP
10001003    8B45 08         MOV EAX,DWORD PTR SS:[EBP+8]
10001006    0345 0C         ADD EAX,DWORD PTR SS:[EBP+0C]
10001009    5D              POP EBP
1000100A    C3              RETN

10001010    55              PUSH EBP
10001011    8BEC            MOV EBP,ESP
10001013    8B45 08         MOV EAX,DWORD PTR SS:[EBP+8]
10001016    2B45 0C         SUB EAX,DWORD PTR SS:[EBP+0C]
10001019    5D              POP EBP
1000101A    C3              RETN

10001020    55              PUSH EBP
10001021    8BEC            MOV EBP,ESP
10001023    8B45 08         MOV EAX,DWORD PTR SS:[EBP+8]
10001026    0FAF45 0C       IMUL EAX,DWORD PTR SS:[EBP+0C]
1000102A    5D              POP EBP
1000102B    C3              RETN

10001030     55             PUSH EBP
10001031     8BEC           MOV EBP,ESP
10001033     8B45 08        MOV EAX,DWORD PTR SS:[EBP+8]
10001036     50             PUSH EAX
10001037     68 00C00010    PUSH OFFSET 1000C000          ; ASCII "Printf from DLL %d "
1000103C     E8 7F000000    CALL 100010C0
10001041     83C4 08        ADD ESP,8
10001044     5D             POP EBP
10001045     C3             RETN
```

Importált függvények címének feloldása

- A program a dll-ben lévő **Add**() függvényt a 00401008 címen a **CALL** [00402000] utasítással hívja meg. Az indirekten hívott függvény címét az Import Address Table (IAT) tárolja.
- A fordító a __declspec(dllimport) **int Add**(**int**,**int**) *;* függvény *tárolási osztály módosító* hatására generál optimálisabb kódot. Ha nem adjuk meg, akkor a **CALL** XXXXXXXX utasításnak meg kellene hívni egy thunk kódot, ami megoldja a DLL-ben lévő függvény hívását. A fordító egy __imp__Add szimbólumot is generál, amely egy pointer, és az IAT táblában foglal helyet.
- A loadernek a betöltésnél csak az IAT-t kell megváltoztatni.
- A függvényeket nemcsak név, hanem sorszám (ordinal number) alapján is meg lehet hívni.

Optimalizálatlan DLL függvényhívás

Ha a főprogramban nem használjuk a __declspec(dllimport) függvény *tárolási osztály módosítót*, ekkor a következő nem annyira hatékony kód keletkezik:

```
00401000   55              PUSH EBP
00401001   8BEC            MOV EBP,ESP
00401003   51              PUSH ECX
00401004   6A 04           PUSH 4
00401006   6A 03           PUSH 3
00401008   E8 21000000     CALL <JMP.&calc.Add>          ; Jump to calc.Add
0040100D   83C4 08         ADD ESP,8
00401010   8945 FC         MOV DWORD PTR SS:[LOCAL.1],EAX
00401013   8B45 FC         MOV EAX,DWORD PTR SS:[LOCAL.1]
00401016   50              PUSH EAX
00401017   E8 0C000000     CALL <JMP.&calc.Function>     ; Jump to calc.Function
0040101C   83C4 04         ADD ESP,4
0040101F   B8 01000000     MOV EAX,1
00401024   8BE5            MOV ESP,EBP
00401026   5D              POP EBP
00401027   C3              RETN
00401028   $- FF25 04204000 JMP DWORD PTR DS:[<&calc.Function>]
0040102E   $- FF25 00204000 JMP DWORD PTR DS:[<&calc.Add>]
```

BIND utility

```
Bind.Exe -v -u test_implicit.exe
BIND: test_implicit.exe - Imports from calc.dll
BIND: test_implicit.exe - Add Bound to 0000000010001000
BIND: test_implicit.exe - Function Bound to 0000000010001030
BIND: Details of binding of test_implicit.exe
    Import from calc.dll [50a80afb]
```

```
00000600: 00 10 00 10 30 10 00 10 00 00 00 00 34 20 00 00  ....0.......4 ..
00000610: FF FF FF FF FF FF FF FF 52 20 00 00 00 20 00 00  ........R .... ..
00000620: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000630: 00 00 00 00 4C 20 00 00 40 20 00 00 00 00 00 00  ....L ..@ ......
00000640: 01 00 46 75 6E 63 74 69 6F 6E 00 00 00 00 41 64  ..Function....Ad
00000650: 64 00 63 61 6C 63 2E 64 6C 6C 00 00 00 00 00 00  d.calc.dll......
```

A Bind utility feloldja az IAT táblában levő címeket. Hogyan lehetséges ez?

- A program relokálása a virtuális memóriakezelés által hardware szintű támogatást kap.
- Az időbélyeg és az Image Bound Import Descriptor (lásd az irodalmat) alapján a loader megbízhatóan el tudja dönteni, hogy a DLL fájl újra lett linkelve.

Irodalom

1. Matt Pietrek An In-Depth Look into the Win32 Portable Executable File Format `http://msdn.microsoft.com/en-us/magazine/bb985992.aspx`
2. Matt Pietrek Peering Inside the PE: A Tour of the Win32 Portable Executable File Format `http://msdn.microsoft.com/en-us/magazine/ms809762.aspx`
3. Microsoft PE and COFF Specification. `http://msdn.microsoft.com/en-us/library/windows/hardware/gg463119.aspx`

Section 1

IDA bevezető

IDA decompiler

- Program szerzője Ilfak Guilfanov, ma már többen dolgoznak rajta.
- Honlap: `http://www.hex-rays.com/`
- IDA demo illetve freeware letölthető a `http://www.hex-rays.com/products/ida/support/download.shtml` helyről.

Objektumorientált kód visszafejtése

Első feladat a főprogram megkeresése.

Mit tudunk:

- A **int** main(**int** argc, char∗argv[], char∗ env[]) program 3 paramétert vesz át.
- 2. paraméter a parancssori argumentum. A Visual Studio-val fordított kódban a mainCRTStartup kód hívja meg a main függvényt, amit a C:\Program Files\Microsoft Visual-Studio 9.0\VC\crt\src\ crt0.c-hez hasonló directoryban találunk meg, a verziószámtól függően.
- A parancssori argumentumokat a GetCommandLine Windows API függvénnyel érjük el.

main program kikeresése

- IDA-ba válasszuk ki <Search> menü alatt <Text> menüpontot, és keressük meg a GetCommandLine függvényt. Mivel ez DLL-ben lévő exportált függvény, ezért a hívó Import Name Table adatában megtalálható a függvény szimbolikus neve, ugyanakkor a main szimbolikus név már kikerült a lefordított kódból.
- Keressünk a függvényhívás közelébe olyan függvényt, aminek három paramétere van.

```
.text:004013F6        mov     eax, dword_40DF78
.text:004013FB        mov     dword_40DF7C, eax
.text:00401400        push    eax
.text:00401401        push    dword_40DF70
.text:00401407        push    dword_40DF6C
.text:0040140D        call    sub_401000
```

Navigálás és átnevezés

- Egér gomb dupla klikk-kel a sub_401000 függvény nevére kattintva a függvény definíciójához ugorhatunk. Ez *escape* gomb hatására visszatérhetünk az előző programrészhez.
- sub_401000 **proc near** *; CODE XREF: start−5C* sorra állva nyomjuk meg az N billentyűt, vagy a jobb egérgomb által előhívott menüből válasszuk ki a <Rename> almenüt, és nevezzük át a függvényt. Térjünk vissza a hívó részhez. Ezt megtehetjük a CODE XREF keresztreferencia részben megadott hívási helyek (jelenleg csak egy) egyikére kattintva. Az adott címen már a **call** main utasítás szerepel. Az IDA a szimbolikus neveket továbbterjeszti, ameddig a program alapján követhető a rá való hivatkozás.

Függvény prototípusának a megadása

Térjünk vissza a main függvényhez, és a jobb egérgomb <Set function type> vagy az Y billentyűvel adjuk meg a függvény prototípusát.

```
int __cdecl main(int argc, char* argv[], char* env[] );
```

Eredmény:

```
.text:00401000 ; int __cdecl main(int argc,char **argv,char **env)
.text:00401000 main    proc near               ; CODE XREF: start-5C
.text:00401000 var_4   = dword ptr -4
.text:00401000 argc    = dword ptr  8
.text:00401000 argv    = dword ptr  0Ch
.text:00401000 env     = dword ptr  10h

.text:00401024          cmp     [ebp+argc], 2
```

Valamint a hívott helyen:

```
.text:00401400          push    eax             ; env
.text:00401401          push    argv            ; argv
.text:00401407          push    argc            ; argc
.text:0040140D          call    main
```

Struktúra (class) definiálás

1. Nyissuk meg a *Structures* dialogus ablakot a shift-F9 gombbal, vagy a <View> <Open subview> <Structures> menü alapján.
2. Hozzunk létre új struktúrát az Ins gombbal, és a *Create structure* ablakban nevezzük el az új struktúrát.
3. Legyen a neve C_class. Az OK gomb megnyomása után létrejön egy üres struktúra.

```
00000000 C_class        struc ; (sizeof=0x0)
00000000 C_class        ends
```

Struktúra tagváltozóinak a megadása

1. A struktúra végére állva a D billentyűvel vehetünk fel új tagváltozókat. Ismételt nyomkodásával az adat méretét változtathatjuk meg. Ha nem találjuk meg a megfelelő méretet, akkor az <Option> <Setup datatype> vagy a Alt-D billentyű lenyomásával hívhatunk elő egy ablakot, amiben be lehet állítani nagyobb méretű változók megjelenését is.
2. Az N gombbal átnevezhetjük a tagváltozókat.

```
00000000 C_class          struc ; (sizeof=0x14)
00000000 c_vtptr          dd ?
00000004 a                dd ?
00000008 b_thunk_vtptr    dd ?
0000000C b                dd ?
00000010 c                dd ?
00000014 C_class          ends
```

Struktúra memóriaterülethez rendelése

1. A var_24 stack frame tartalmazza a C c*;* objektumot. Az adott értékre duplán kattintva bejön a *Stack frame* ablak.
2. Az <Edit> <Struct var> menük, vagy az Alt-Q billentyű segítségével hívjuk be a *"choose a structure"* ablakot és válasszuk ki a megfelelő struktúra definíciót.

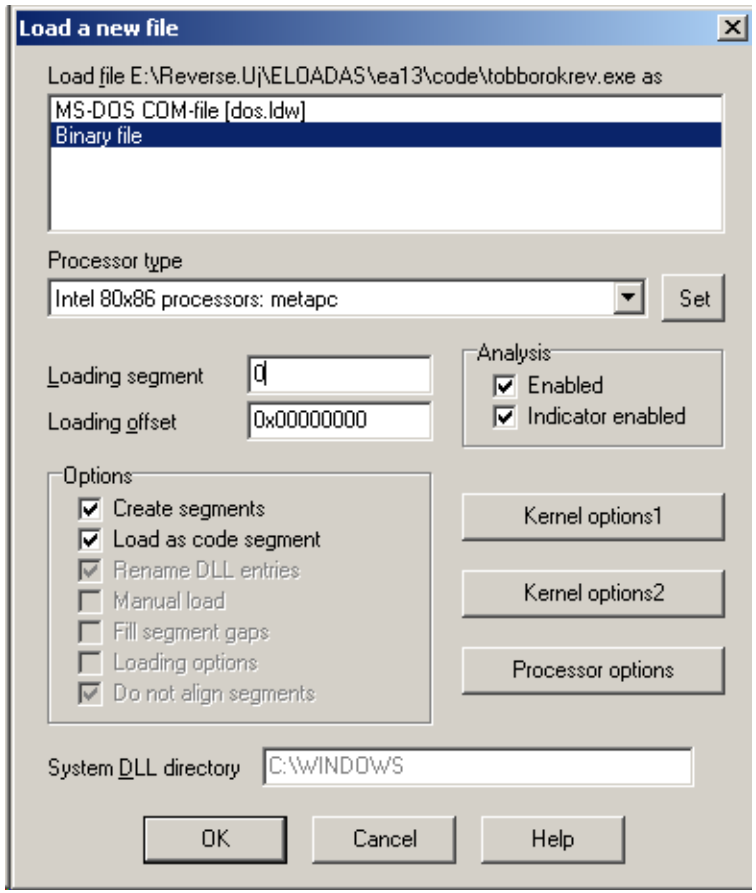
```
.text:00401000 main     proc near    ; CODE XREF: start-5C
.text:00401000
.text:00401000 var_28   = dword ptr -28h
.text:00401000 c        = C_class ptr -24h
.text:00401000 var_10   = dword ptr -10h
.text:00401000 var_C    = dword ptr -0Ch
.text:00401000 var_4    = C_class ptr -4
```

Section 2

IDA

Bináris fájlformátum visszafejtése

Az egyszerűség kedvéért nevezzük át egy PE programban lévő MZ szignatúrát RE-re. Ekkor az IDA nem ismeri fel, így bináris fájlként tudjuk beolvasni.



Struktúrák definiálása

Definiáljuk a következő struktúrákat az IDA-ban (Shift+F9):

- DOS header
- NT header
- Section header

Dos header definiálása

seg000:00000000 címhez rendeljük hozzá az <Edit> <Struct var> vagy Alt-Q gombbal a dos_header struktúrát.

```
seg000:00000000     dos_header <4552h, 90h, 3, 0, 4, 0, 0FFFFh, 0, 0B8h,\
seg000:00000000                0, 0, 0, 40h, 0, 0, 0, 0, 0, 0D8h>
seg000:00000040     db  0Eh
```

Az e_lfanew tagváltozó értéke 0xD8, ami a fájl elejétől mérve az NT header kezdete.

NT header definiálása

seg000:000000D8 címhez rendeljük hozzá az <Edit> <Struct var> vagy Alt-Q gombbal a nt_header struktúrát.

```
seg000:000000D8  nt_header <4550h, 14Ch, 3, 50B3F04Ch, 0, 0, 0E0h, \
seg000:000000D8                103h, 10Bh, 9, 0, 8C00h, 5400h, 0, 1469h,\
seg000:000000D8                1000h, 0A000h, 400000h, 1000h, 200h>
seg000:00000118  db     5
```

Az OptionalHeader az NT headerben a 0x18-cal eltolva kezdődik, és a mérete 0xe0. Ezért az első szegmens header tábla a 0xd8+0x18+0xe0 = 0x1d0 helyen kezdődik. A szegmensek száma 3.

Section header definiálása

seg000:000001d0 címhez rendeljük hozzá az <Edit> <Struct var> vagy Alt-Q gombbal a section_header struktúrát. Mivel 3 section van, ezért a műveletet ismételjük meg még kétszer.

```
seg000:000001D0  section_header <'.text', 8B04h, 1000h, 8C00h, 400h, \
seg000:000001D0                  0, 0, 0, 0, 60000020h>
seg000:000001F8  section_header <'.rdata', 26B4h, 0A000h, 2800h, 9000h, \
seg000:000001F8                  0,  0, 0, 0, 40000040h>
seg000:00000220  section_header <'.data', 2B48h, 0D000h, 1000h, 0B800h, \
seg000:00000220                   0, 0, 0, 0, 0C0000040h>
seg000:00000248      db     0
```

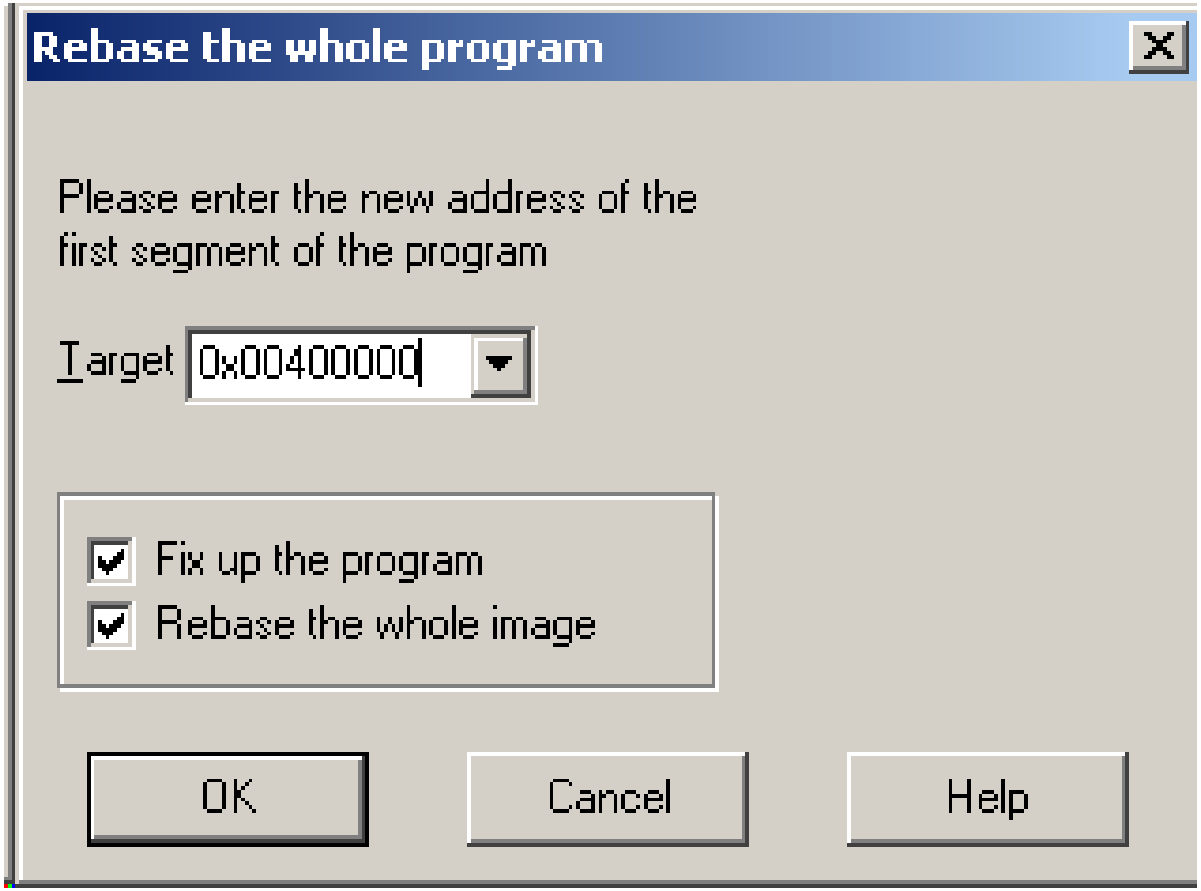
Memóriabeli helyek meghatározása

A programot a 0 címtől kezdve töltöttük be, és a section-ok sem 4096 byte többszörösén, azaz nem laphatáron kezdődnek.

- Program kezdete: OptionalHeader.BaseOfImage 0x00400000.
- .text section
  - kezdete a fájlban PointereToRawData = 0x400
  - mérete a fájlban SizeOfRawData = 0x8c00
  - memóriában a helye RVA-ban VirtualAddress 0x1000
- .rdata section
  - kezdete a fájlban PointereToRawData = 0x9000
  - mérete a fájlban SizeOfRawData = 0x2800
  - memóriában a helye RVA-ban VirtualAddress 0x0A000
- .data section
  - kezdete a fájlban PointereToRawData = 0x0b800
  - mérete a fájlban SizeOfRawData = 0x1000
  - memóriában a helye RVA-ban VirtualAddress 0x0D000

Program betöltési helyének megváltoztatása

<Edit> <Rebase program...>



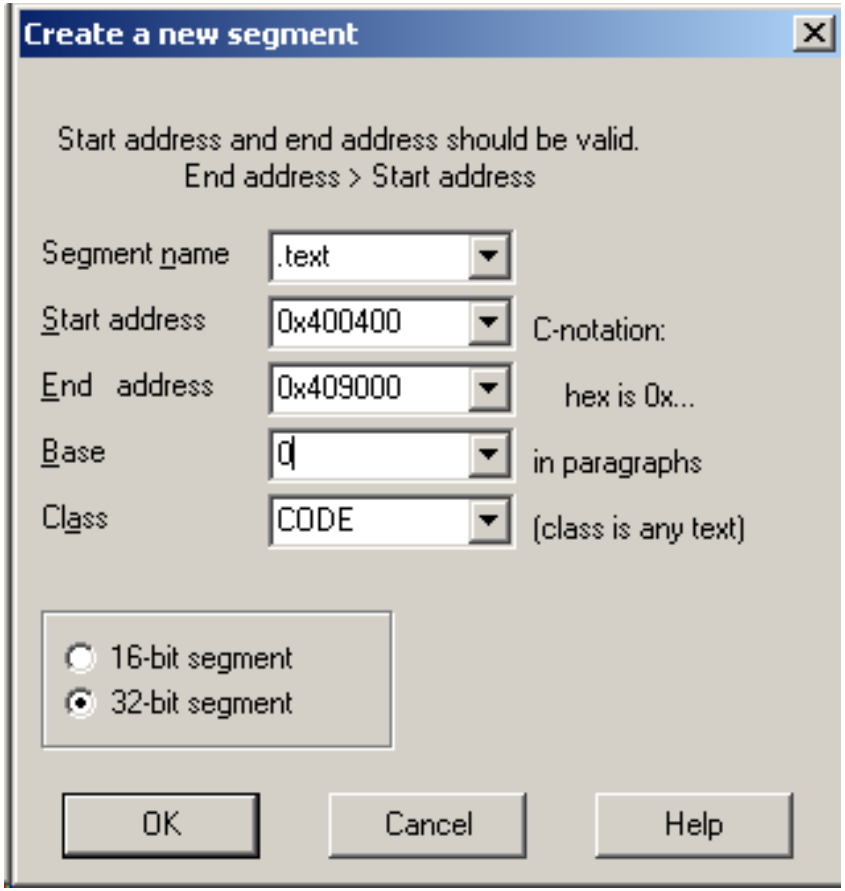
Új 3 segment létrehozása

Szegmensek kezdete és vége:

- .text 0x00400400-0x00409000
- .rdata 0x00409000-0x0040B800
- .data 0x0040B800-0x0040C800

Segment-ek létrehozása

Új szegmenseket a <Edit> <Create new segment> menüpont kiválasztásával lehet létrehozni.



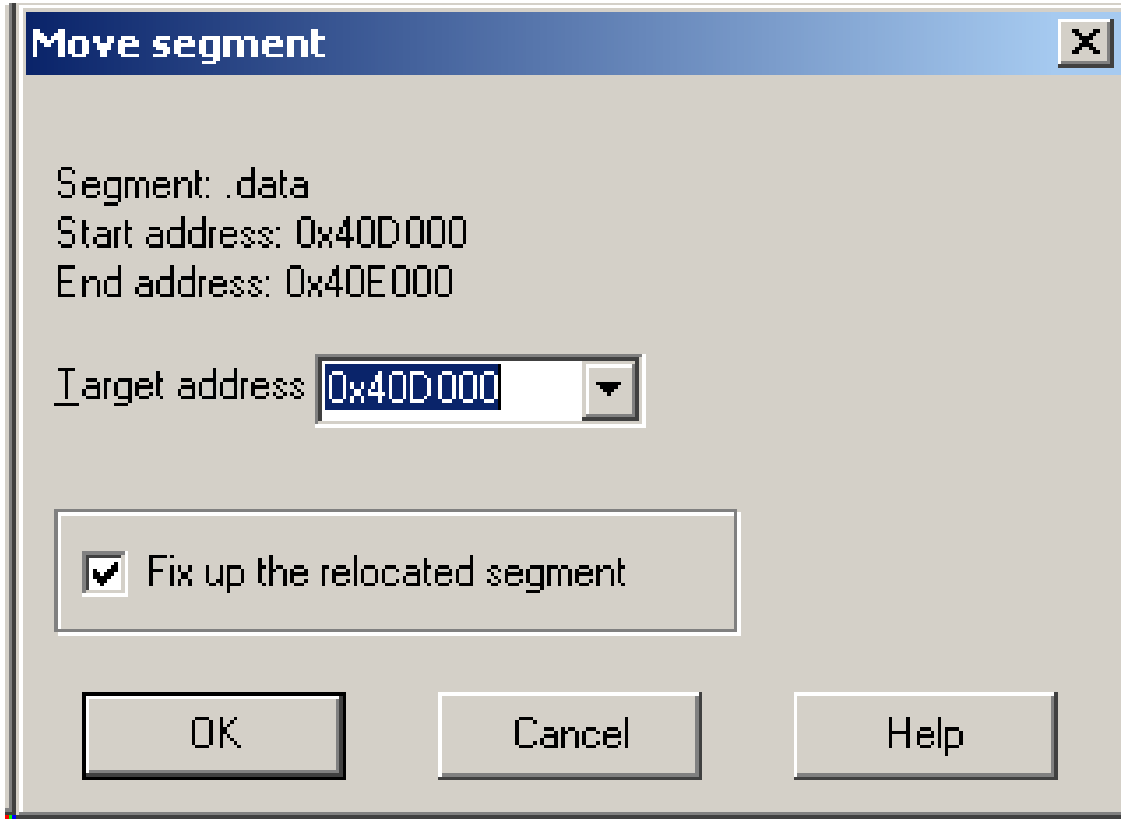
Section-ok elhelyezkedése

<View> <Open subviews> <Segments> vagy Shift-F7 gombbal megnézhetjük a definiált szegmenseket .

seg000	00400000	00400400	?	?	?	.	L	**byte**	0001	**public**	**CODE**	32	0000
.text	00400400	00409000	?	?	?	.	.	**byte**	0000	**public**	**CODE**	32	FFFFFFFF
.rdata	00409000	0040B800	?	?	?	.	.	**byte**	0000	**public**	RDATA	32	FFFFFFFF
.data	0040B800	0040C800	?	?	?	.	.	**byte**	0000	**public**	DATA	32	FFFFFFFF

Szegmensek eltolása

Érdemes a legnagyobb című szegmenstől visszafele haladni, és az <Edit> <Move current segment> paranccsal a szegmens headerben megadott Virtual Address, és a program kezdőcímének megfelelően, az IDA adatbázisban lévő adatokhoz új címet rendelni.



Kód visszafejtés

NT header Opcional Header részében az Address of Entry Point értéke 0x1469h. Ehhez hozzáadva a program kezdőcímét (0x400000) megkapjuk az első végrehajtandó utasítás címét. Ettől a címtől kell az <Edit> <Code> vagy a C betű segítségével a kódot visszafejteni.

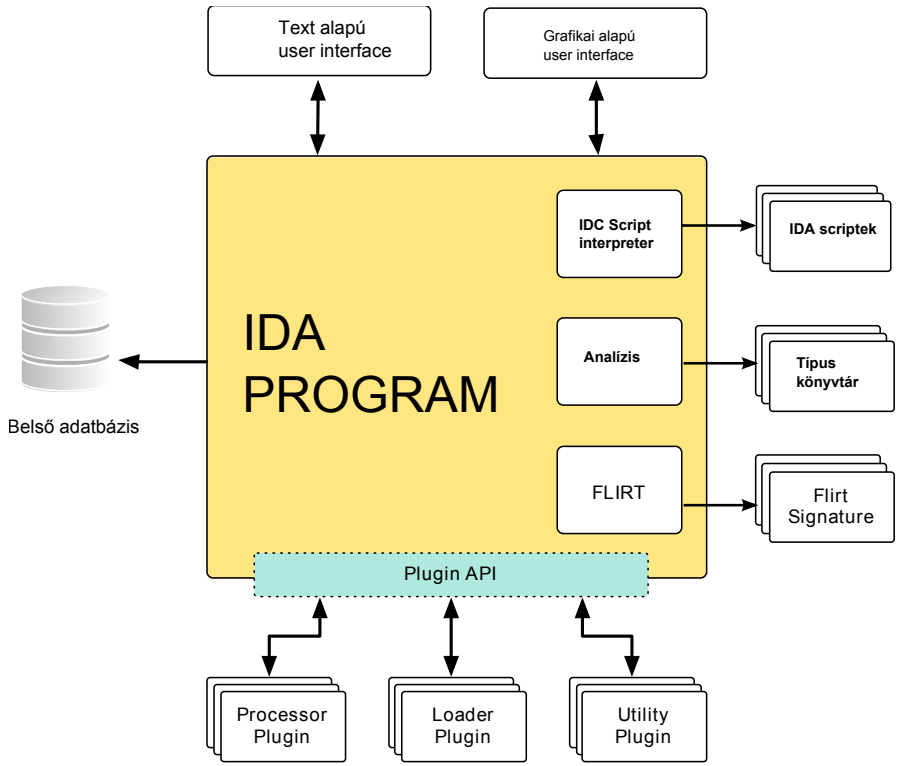
A kód minősége rosszabb, mint az automatikus visszafejtés esetén.

IDA programozási lehetőségek

- IDA script (nem elég rugalmas).
- IDA plugin rugalmas, de ismerni kell a C++ nyelvet. Az IDA API függvényei a következő feladatokat támogatják:
  - Fájl formátum felismerése és leképzése az IDA adatbázisba.
  - Különböző processzorok kódjának a disassemblálása.
  - Adat és vezérlésszerkezet analízis támogatása.
  - Grafikus megjelenítés a kódrészek kiemelésére, valamit szemléltető ábra, gráf készítés .
  - Saját analízishez kényelmes felhasználói interface létrehozása (Qt toolkit).
  - Új processzor utasításainak emulálásához szükséges rutinok létrehozása. Ezt a debugger modul fogja használni.
  - És még sok egyéb, talán még most nem is ismert funkciók létrehozása.

IDA plugin fajtái

- utility plugin
- loader plugin
- processzor plugin



IDA plugin élettartama

- IDA adatbázis betöltésétől a kilépésig;
- betöltő (loader) modul működési idejéig;
- processzor (visszafejtő) modul működési idejéig;
- igény szerint, interaktívan, vagy callback.

IDA Plugin

```
plugin_t __declspec(dllexport) PLUGIN =
{
  IDP_INTERFACE_VERSION,
  0,                    // plugin flags
  init,                 // inicializálás
  term,                 // terminálás. a fv  pointer lehet NULL is.
  run,                  // plugin rutin lefuttatás
  comment,              // plugin leírása
                        // látható a status sorban
                        // vagy tippként
  help,                 // többsoros leírása a plugin-ról
  wanted_name,          // plugin neve
  wanted_hotkey         // hotkey a plugin futtatásához
};
```

Plugin inicializálás

Ellenőrzés, hogy a plugin a megnyitott IDA adatbázishoz alkalmazható-e.

- Adott fájlformátumhoz készült.
- Adott CPU-t támogatja.
- IDA verzió megegyezik.
- GUI vagy text módban fut.
- Használ más plugint, és az be van-e töltve.
- és egyéb szempontok

```
int idaapi init(void)
{
    if( is_plugin_compatible_with_actual_database() )
    {
        return PLUGIN_OK; // usr activated
    }
    return PLUGIN_SKIP; // unload
}
```

Felhasználói Plugin aktiválás

- Hagyományos módon <Edit> <Plugin> <MyPlugin> <calls run()>
- Új menüpont hozzáadásával a (add_menu_item()) függvény segítségével.

```
inline bool add_menu_item(
        const char *menupath,
        const char *name,
        const char *hotkey,
        int flags,
        menu_item_callback_t* callback,
        void* ud ) ;

typedf bool idaapi menu_item_callback_t(void* ud);
```

Plugin aktiválás események bekövetkezésére

Callback függvény regisztrálása:

```
// preprocesszor eseményre
hook_to_notification_point( HT_IDP, idp_callback, NULL);

// adatbázis változásra
hook_to_notification_point( HT_IDB, idb_callback, NULL);

// felhasználói interfész esetén
hook_to_notification_point( HT_UI, ui_callback, NULL);

// debugger eseményre
hook_to_notification_point( HT_DBG, dbg_callback, NULL);

typedf bool idaapi menu_item_callback_t(void* ud);
```

Új IDC függvény létrehozás

```
idc_func("MyFunc5", myfunc5, myfunc5_args);
```

Loader plugin létrehozás bináris fájl értelmezésére

Mire jó? Végtelen sok futtatható fájlformátum lehet.

- ROM-ban lévő kód;
- TCP csomagokban terjedő kódok, amit a tcpdump vagy egyéb programok segítségével fájlba menthetünk;
- önkicsomagoló tömörítő exe programok, amihez nekünk kell elvégezni az image létrehozását a visszafejtett algoritmus alapján;
- önkicsomagoló exe esetén, amikor rábízzuk a kicsomagolást a programra, azaz lefuttatjuk, és a memóriában keletkező kódot (processzhez tartozó lapokat) fájlba mentjük.
- stb.

Loader inicializálása

Előző diákon bemutatott, kézzel visszafejtett RE szignatúrával rendelkező, de ténylegesen PE fájlformátumhoz készítjük el a loader-t.

```
//
// Loader Module Descriptor Blocks
//
extern "C" loader_t LDSC = {
  IDP_INTERFACE_VERSION,
  0, /* no loader flags */
  accept_file,
  load_file,
  save_file,
  NULL,
  NULL,
};
```

accept_file

A függvény megnézi, hogy a fájl elején megtalálható-e az "RE" string.

```
int __stdcall accept_file(
        linput_t *li,
        char fileformatname[MAX_FILE_FORMAT_NAME],
        int n)
{
    uint32 magic;
    if( n || lread4bytes( li, &magic, false) )
         return 0;

    if (magic != RE_MAGIC ) return 0;

    // file has passed all
    qstrncpy(fileformatname, "REVERESE (PE) Image",
        MAX_FILE_FORMAT_NAME);

     // send messagess
     qsnprintf(fileformatname, MAX_FILE_FORMAT_NAME,
                "REVERSE Executable" );

     return ACCEPT_FIRST | 1;
}
```

save_file

Ezt a függvényt az IDA a <File> <Produce file> <Create exe file...> menü aktiválása esetén hívja meg. Célja az adatbázis alapján előállítani a futtatható fájt, de a kiírt adat a programozóra van bízva.

```
// Demo only
int idaapi save_file( FILE* fp,
      const char* fileformatname )
{
   qfwrite(fp, fileformatname, strlen(fileformatname) );
   // databese -> exec file
   // base2file(...);
   return 1;
}
```

Típusok megadása

Az 5.0 verziójú IDA nem ismeri a parse_decls() függvényt, ezért az IDA-ban definiált struktúra típust használtam, amit az IDA .til kiterjesztésű fájlban tárol. Ez a fájl csak akkor érhető el, amikor az IDA aktív, különben a .idb-be összecsomagolja a fájlokat. A .til fájlt az <IDA DIR>\til directory-ba kell másolni.

```
tid_t    dos_header_struct;
tid_t    nt_header_struct;
tid_t    opt_header_struct;
tid_t    section_header_struct;

void add_types()
{
    add_til( "reexe.til");   // til fájl olvasás
    dos_header_struct = til2idb(-1, "dos_header");
    nt_header_struct = til2idb(-1, "nt_header");
    opt_header_struct = til2idb(-1, "optional_header");
    section_header_struct = til2idb(-1, "section_header");
}
```

Header-ek beolvasás

A load_file() elsőnek a DOS header-t olvassa be, de a többi header struktúra beolvasása is hasonlóan történik.

```
void __stdcall load_file(
    linput_t *li,
    ushort /*neflag*/, const char * /*fileformatname*/)
{
    struct dos_header dhdr;
    uint32 pos=0
    add_types();
    // DOS header feldolgozás
    lread(li, &dhdr, sizeof( struct dos_header ) );
    mem2base( &dhdr, pos, pos+ sizeof( struct dos_header ), pos );

    if( !add_segm(0, pos, pos+sizeof( struct dos_header ), ".doshdr",
                  "DATA" ) ) {
       loader_failure();
    }
    segment_t *dos_seg = getseg( 0 );
    set_segm_addressing(dos_seg, 0 );  // 16 bit
    doStruct(pos, sizeof( struct dos_header ), dos_header_struct );
    ...
```

IDA függvények

- lread() fájlból adatokat olvas a memóriába.
- mem2base() memóriában lévő adatokat az IDA adatbázisba tölti.
- file2base() a fájlban lévő adatokat az IDA adatbázisba tölti.
- add_seg() szegmens létrehozás, interaktívan a <Create new segment> paranccsal adható meg.
- getseg() a paraméterként átadott lineáris cím alapján visszaadja az IDA adatbázisban lévő szegmens leírót.
- set_segm_addressing() Szegmens címzési módja 16, 32 vagy 64 bites. Kód visszafejtésénél nagyon fontos megadni.
- doStruct() Adatok struktúra típushoz rendelése, interaktívan az ALT-Q gombbal lehet megadni.

Fájban lévő szegmensek adatbázisba töltése

Kis DOS-os programocska betöltése.

```
pos += sizeof( struct dos_header ); // file pos léptetés

file2base( li , pos, pos, dhdr.e_lfanew, FILEREG_PATCHABLE );
if( !add_segm(0, pos, dhdr.e_lfanew, ".dosprg", "DATA" ) ) {
    loader_failure();
}
segment_t *dos_prg = getseg( pos );
set_segm_addressing(dos_prg, 0 );
pos = dhdr.e_lfanew;

...
```

Szegmensek mozgatása

Szegmensek laphatárra igazítása. Interaktívan az <Edit> <Move current segment> menüpont alatt érhetjük el ezt a funkciót.

```
for(int i=nthdr.NumberOfSection-1; i>= 0; i--)
{
    segment_t *s = getseg( pSec[i].PointereToRawData );
    move_segm( s, pSec[i].VirtualAddress, 0 );
}
```

Teljes program áthelyezése

Mivel az exe file a 0x400000 címhez képest tartalmazza a kódban a címeket, ezért az adatbázisba lévő kódot is át kell helyezni erre a címre. Program betöltési helyének megváltoztatása interaktívan az <Edit> <Rebase program...> menü segítségével történt.

```
// Rebase the whole program by 'delta' bytes
rebase_program( opthdr.BaseOfImage, 0 );

...
```

Belépési pont definiálás

Meg kell adni az első végrehajtandó utasítás címét, mert az IDA disassembler ettől a címtől kezdi visszafejteni a programot.

```
// Rebase the whole program by 'delta' bytes

// add entry point
add_entry(opthdr.AddressOfEntryPoint + opthdr.BaseOfImage,
        opthdr.AddressOfEntryPoint+opthdr.BaseOfImage,
        "_start", true );

// This function should be called only from the loader
//  to describe the input file.
create_filename_cmt();
...
```

Plugin installálás

A lefordított plugint a megfelelő IDA directory alá kell másolni.

- <IDA DIR>\plugin\ általános célú tool-ok
- <IDA DIR>\loaders\ fájlformátum értelmező plugin-ek
- <IDA DIR>\procs\ processzor assembly kódját visszafejtő plugin

Section 3

PIN

Pin - A Dynamic Binary Instrumentation Tool

homepage:
`http://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool`

Bináris Instrumentation

Instrument – műszer, szerszám,
bináris instrumentation – kód mérése.

Előnyök:

- nyelvfüggetlen;
- gépi kód szintű;
- forrásfájl nélküli programokhoz is alkalmas

Az instrumentation történhet statikusan – futás előtti kódinjektálás, vagy dinamikusan – azaz futás közben.
Dinamikus módszer előnye:

- nem kell újrafordítani, újralinkelni;
- futás közben fedezi fel a kódot;
- alkalmas dinamikusan generált kódhoz is;
- futó program közben is lehet alkalmazni.

Néhány felhasználási terület

- kód tesztelésénél a kódlefedés mérésére;
- hívási fa generálás;
- memória szivárgás tesztelésére;
- párhuzamosan futó szálak vizsgálata;
- programrészek futási sebességének a mérése ;
- új gépi utasítások bevezetésének a tesztelése;
- visszafejtés esetén a megoldás ellenőrzése;
- utasítás, függvény lecserélése saját kódra.

PIN-tool futtatása

```
pin  -t  pintool.dll -- application
```

- pin – instrumentation motor;
- pintool.dll – felhasználó által írt instrumentation eszköz;
- application – tetszőlegesen kiválasztott vizsgálandó program;

Futó program instrumentálása

```
pin  -mt 0 -t pintool.dll -pid 1234
```

Instrumetation API részei

- *Instrumentation rutin* – program futása során az első alkalommal, amikor valahova kerül a vezérlés, hozzárendeli azt a rutint, ami minden esetben az utasítás végrehajtásakor le fog futni;
- *Analízis rutin* – az a kód, amit az *instrumentation rutin* hozzárendelt az utasításhoz. Az *instrumentált utasítások* végrehajtása során mindig meghívódik az analízis rutin;

Utasítás számlálás elve

```
counter++
push ebp
counter++
mov ebp,esp
counter++
sub esp,10h
counter++
mov dword ptr [ebp-4],0
counter++
mov eax,dword ptr [ebp+8]
counter++
cmp eax,0
counter++
jz L1
counter++
mov ecx,dword ptr [ebp-4]
counter++
jmp L2
```

Utasítás számlálás PIN tool-lal

```
#include <iostream>
#include "pin.h"

UINT64 icount = 0;

// analízis rutin
void docount() { icount++; }

// instrumentation rutin
void Instruction(INS ins, void *v)
{
    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)docount, IARG_END);
}

void Fini(INT32 code, void *v)
{ std::cerr << "Count " << icount << endl; }

int main(int argc, char * argv[])
{
    PIN_Init(argc, argv);
    INS_AddInstrumentFunction(Instruction, 0);
    PIN_AddFiniFunction(Fini, 0);
    PIN_StartProgram();
    return 0;
}
```

Utasításvégrehajtás nyomkövetésének elve

```
print(EIP)
push ebp
print(EIP)
mov ebp,esp
print(EIP)
sub esp,10h
print(EIP)
mov dword ptr [ebp-4],0
print(EIP)
mov eax,dword ptr [ebp+8]
print(EIP)
cmp eax, 0
print(EIP)
jz L1
print(EIP)
mov ecx,dword ptr [ebp-4]
print(EIP)
jmp L2
```

Utasításvégrehajtás nyomkövetése PIN tool-lal

```
#include <iostream>
#include "pin.h"
FILE * trace;

// analízis rutin
void printip(void *ip) { fprintf(trace, "%p\n", ip); }

// instrumentation rutin
void Instruction(INS ins, void *v)  {
    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)printip,
                   IARG_INST_PTR, IARG_END);

}

void Fini(INT32 code, void *v)  { fclose(trace); }

int main(int argc, char * argv[]) {
    trace = fopen("itrace.out", "w");
    PIN_Init(argc, argv);
    INS_AddInstrumentFunction(Instruction, 0);

    PIN_AddFiniFunction(Fini, 0);
    PIN_StartProgram();
    return 0;
}
```

Instrumentation finomsága

- *utasítást szintű*
- *alapblokk szintű* Az alapblokk fogalma más mint amit a fordítók elméletében használnak, mivel itt a blokk közepébe is be lehet ugrani. Azaz egy blokknak több belépési pontja is lehet, de csak egy kilépési pontja van. Ez érthető, mivel futás közben nem lehet megállapítani, hogy van-e más helyről is belépés a blokkba.
- *trace szintű* A trace végét feltétel nélküli vezérlésátadó utasítás zárja pl. **jmp** vagy **ret**.

```
sub     edx,0FFh
cmp     edx, esi
jle     L1
```

```
mov     edi,1
add     eax, 010h
jmp     L2
```

1 trace
2 Basic Block
6 utasítás

Gyorsabb utasítás számlálás

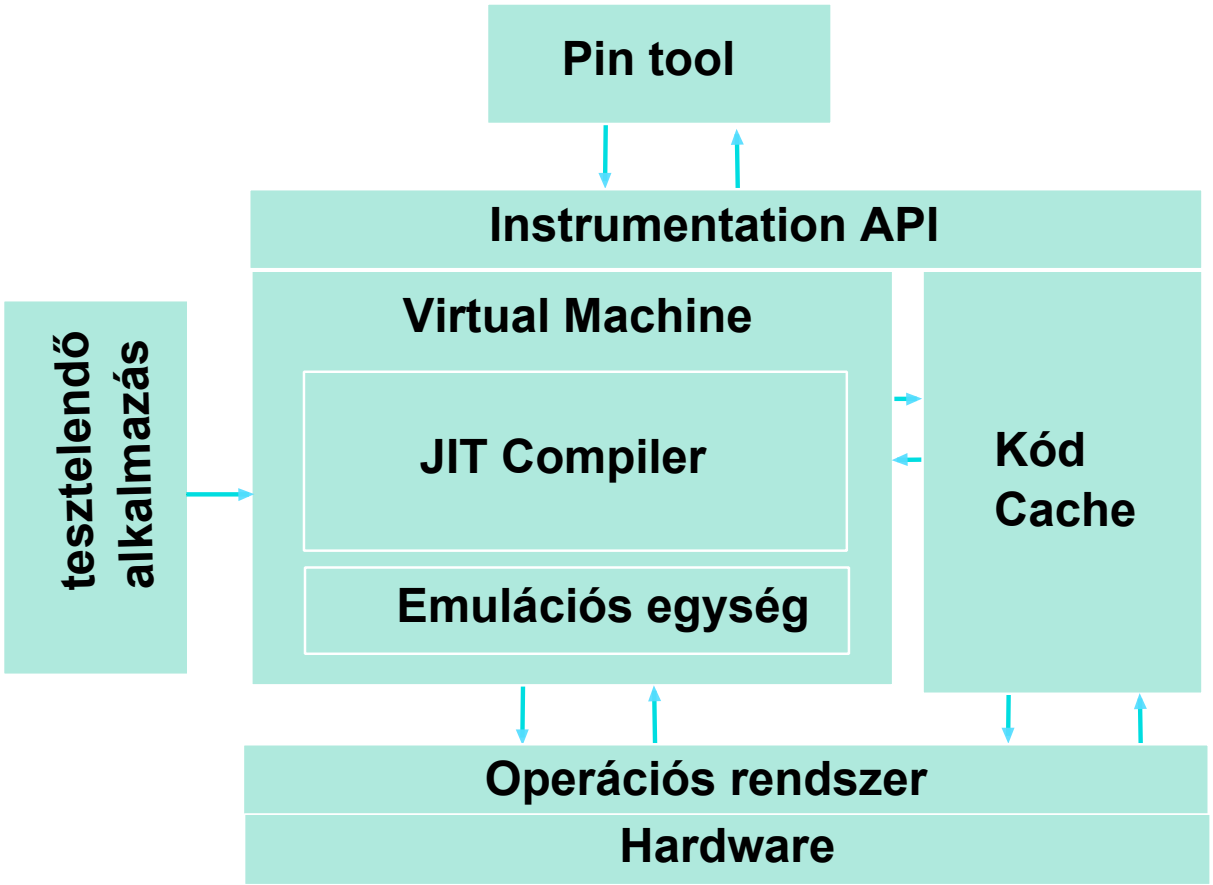
```
counter+=7
push ebp
mov ebp,esp
sub esp,10h
mov dword ptr [ebp-4],0
mov eax,dword ptr [ebp+8]
cmp eax, 0
jz L1
counter+=2
mov ecx,dword ptr [ebp-4]
jmp L2
```

Gyorsabb utasítás számlálás PIN tool-lal

```
#include <stdio.h>
#include "pin.H"
UINT64 icount = 0;
void docount(INT32 c) { icount += c; }

void Trace(TRACE trace, void *v) {
  for (BBL bbl = TRACE_BblHead(trace);
    BBL_Valid(bbl); bbl = BBL_Next(bbl)) {
    BBL_InsertCall(bbl, IPOINT_BEFORE, (AFUNPTR)docount,
          IARG_UINT32, BBL_NumIns(bbl), IARG_END);
  }
}

void Fini(INT32 code, void *v) {
  fprintf(stderr, "Count␣%lld\n", icount);
}

int main(int argc, char * argv[]) {
   PIN_Init(argc, argv);
   TRACE_AddInstrumentFunction(Trace, 0);
   PIN_AddFiniFunction(Fini, 0);
   PIN_StartProgram();
   return 0;
}
```

PIN architektúra

Úgy fogható fel mint egy virtuális gép, csak nem byte kódot, hanem gépi kódot értelmez.



Számított vezérlésátadási címek nyomkövetése

```
#include <stdio.h>
#include "pin.H"
FILE * trace;

VOID DumpJmpAddr(VOID * ip, VOID * addr)
{     fprintf(trace,"%p: %p\n", ip, addr ); }

VOID Instruction(INS ins, VOID *v) {
   if( INS_IsBranchOrCall(ins) && INS_OperandIsReg(ins, 0))      {
      INS_InsertCall(
            ins, IPOINT_BEFORE, (AFUNPTR)DumpJmpAddr,
            IARG_INST_PTR,
            IARG_BRANCH_TARGET_ADDR,
            IARG_END);
    }
}

VOID Fini(INT32 code, VOID *v) {
    fprintf(trace, "#eof\n");
    fclose(trace);
}

INT32 Usage() {
    PIN_ERROR( "This Pintool prints a trace of calculated call\n"
              + KNOB_BASE::StringKnobSummary() + "\n");
    return -1;
}

int main(int argc, char *argv[]) {
```

Dinamikus nyomkövetés eredményének felhasználása

Pin-tool output :

```
004036DA: 7C9132D9
004029CE: 0040157E
004029CE: 00404CE3
004029CE: 00405701
```

Az IDA által visszafejtett kód a PIN -tool által kiírt címen:

```
004029C8        mov     ecx, [esi]
004029CA        test    ecx, ecx
004029CC        jz      short loc_4029D0
004029CE        call    ecx
004029D0
004029D0 loc_4029D0:                    ; CODE XREF: sub_4029B7+15j
```

A hívott rutin, amit a rekurzívan pásztázó disassembler nem tudott feloldani, mert nem ismerte az **ecx** regiszter értékét.

```
0040157D sub_401578     endp
0040157E        db 0A1h ;
0040157F        db  40h ;
00401580        db 0FBh ;
```

Ajánlott olvasmány

Ami már nem fért bele az órába, de érdemes elolvasni:

`http://www.codeproject.com/Articles/30815/An-Anti-Reverse-Engineering-Guide`