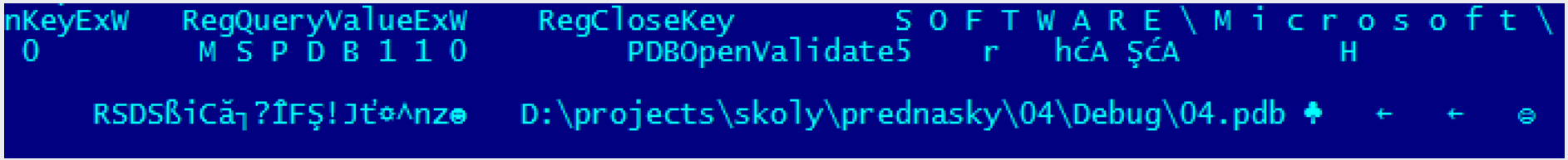
Memory management a PE súbor

filip.kafka@eset.sk

Základy reverzného inžinierstva

Debugging symboly

- Symboly, ktoré pomáhajú pri ladení kódu, obsahujú prototypy, adresy funkcii, adresy premenných
- Dá sa spraviť vlastný symbol server, kde si môžem odkladať symboly svojich release spustiteľných súborov
- Microsoft používa .pdb súbory vo formáte COFF
- GCC namiesto toho používa .dwarf
- srv*c:\Symbols*http://msdl.microsoft.com/download/symbols;
- Prítomnosť .pdb cesty môže prezradiť informácie o autorovi, projekte, ...



Naked funkcie

- Microsoft specific rozšírenie do jazyka C
- Naked funkcie neobsahujú prológ a epilóg (je to nechané na programátorovi)
- Kompilátor počas optimalizácie často vytvára takéto naked funkcie
- Naked funkcia bez epilógu nemá inštrukciu **`ret`**

```
__declspec(naked) void naked_function() {
    printf("Naked function\n");
    __asm{ ret }
}

int main(void) {
    naked_function();
    return 0;
}
```

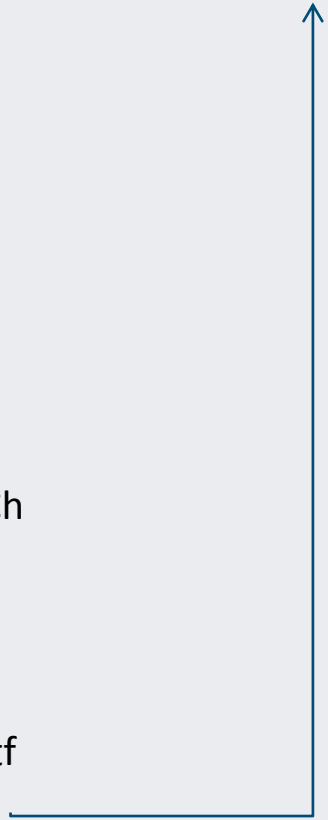
```
; void __cdecl naked_function()
?naked_function@@YAXXZ proc near
push    offset Format   ; "Naked function\n"
call    ds:__imp__printf
add     esp, 4
retn
?naked_function@@YAXXZ endp
```

Entry point = start()

- Miesto, od ktorého sa začne program vykonávať. Inicializuje globálne premenné, volá konštruktory staticky alokovaných premenných, nastavuje SEH
- start() ≠ main()
- main() – implementuje len našu funkcionalitu, zavolá sa po spustení aplikácie (.exe), po návrate z neho proces končí
  Zavolaniu funkcie main() môže predchádzať mnoho inicializačných rutín: _mainCRTStartup(), ___dyn_tls_init_callback(), initterm_e(), ...
- WinMain() – Windows specific, robí to isté čo main
- DllMain() – callback, volaný pri load/unload do procesu alebo vytvorení/ukončení vlákna

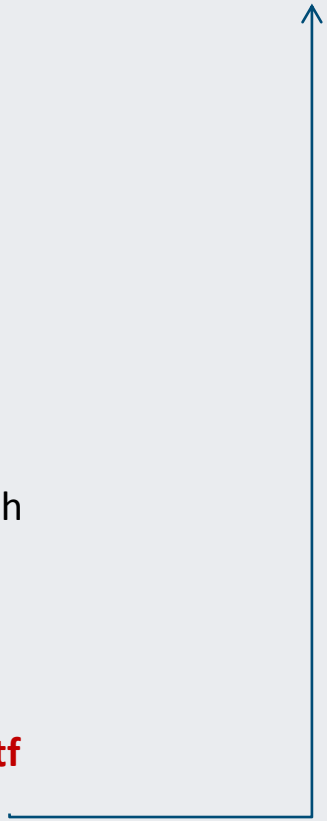
Čo robí nasledovný kód?

```
push    ebp
mov     ebp, esp
sub     esp, 0C0h
push    ebx
push    esi
push    edi
lea     edi, [ebp-0C0h]
mov     ecx, 30h
mov     eax, 0CCCCCCCCh
rep stosd
mov     esi, esp
push    offsetFormat
call    ds:__imp__printf
add     esp, 4
cmp     esi, esp
call    j___RTC_CheckEsp
xor     eax, eax
pop     edi
pop     esi
pop     ebx
add     esp, 0C0h
cmp     ebp, esp
call    j___RTC_CheckEsp
mov     esp, ebp
pop     ebp
ret
```



Čo robí nasledovný kód?

```
push    ebp
mov     ebp, esp
sub     esp, 0C0h
push    ebx
push    esi
push    edi
lea     edi, [ebp-0C0h]
mov     ecx, 30h
mov     eax, 0CCCCCCCCh
rep stosd
mov     esi, esp
push    offsetFormat
call    ds:__imp__printf
add     esp, 4
cmp     esi, esp
call    j___RTC_CheckEsp
xor     eax, eax
pop     edi
pop     esi
pop     ebx
add     esp, 0C0h
cmp     ebp, esp
call    j___RTC_CheckEsp
mov     esp, ebp
pop     ebp
ret
```



Objekt a handle

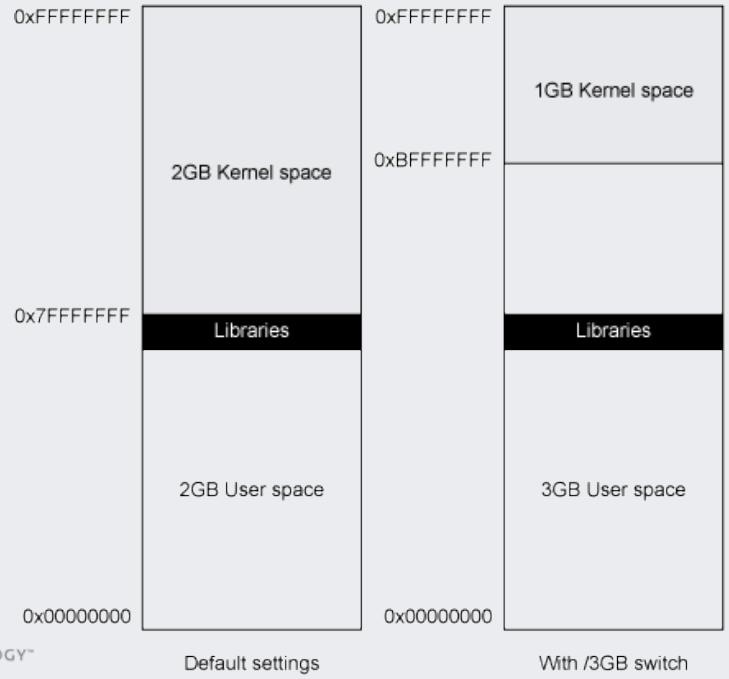
- Objekt je dátová štruktúra, ktorá reprezentuje systémové zdroje (súbor, vlákno, okno, ...)
- Všetky objekty majú rovnakú hlavičku, ktorá ich popisuje (typ, meno, počet referencií, ...)
- Objekty sú spravované systémom – Object manager (SI - WinObj)
- Handle predstavuje Windows objekt z pohľadu užívateľského priestoru. K objektom sa nedá pristupovať priamo, len nepriamo cez handle
- Handle je len indexom v Process handle table (SI - handle)
- Process handle table obsahuje takisto masku oprávnení, ktoré definujú, aké operácie je možné s daným objektom pre dané PID vykonávať

Relative Virtual Address (RVA), Position independent code (PIC)

- Relatívne adresovanie
  - Adresovanie vzhľadom na určitú bázu (začiatok sekcie, začiatok súboru, začiatok tabuľky, ...)
  - RVA: vzhľadom na ImageBase
  - Relatívne adresovanie môže byť kratšie
  - V určitých situáciách jednoduchšie použiteľné
- PIC:
  - JMP (EB _) - short jump
  - JMP (E9 _ _ _ _) - long jump
  - CALL (E8 _ _ _ _)
  - ...

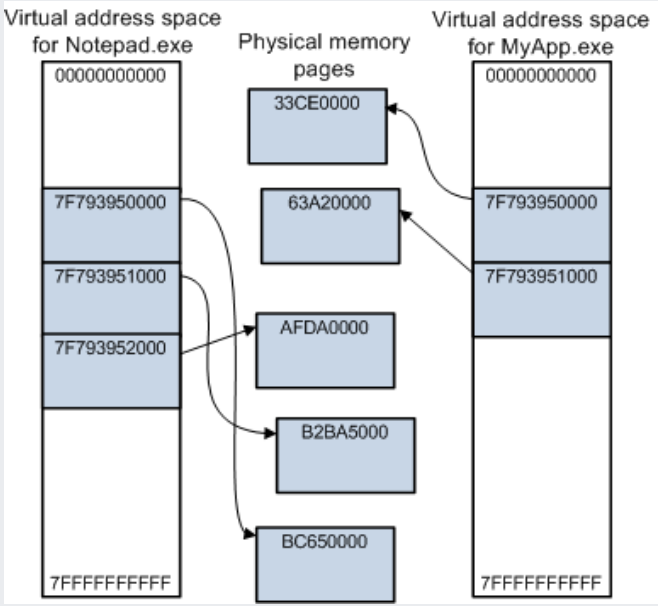
Separovanie programov podľa privilégii

- Väčšina 32-bit procesorov poskytuje ring0 (kernel) – ring3 (user) mód
- Väčšina OS (Windows, Linux) využíva len ring 0 a ring 3
- User mode programy môžu „vyžiadať“ kernel mode operáciu od OS



Memory management

- Fyzické vs virtuálne adresy (na preklad slúži register CR3)
  Real mode (len pri bootovaní) vs protected mode



Stránkovanie

- Stránkovanie – rozdelenie pamäti na rovnako veľké stránky s možnosťou nastavenia atribútov (read, write, execute)

  - bezpečnosť (separovanie procesov)

  - efektívnosť (ukladanie nepoužívaných stránok na disk - swapovanie)

- Segmentácia – používaná sa na spohodlnenie práce

Proces & vlákno

- O ich prepínanie sa stará OS
- Proces:
  - Inštancia programu, ktorá sa vykonáva
  - Má vlastný adresný priestor
- Vlákno:
  - Základná jednotka vykonávania CPU jadra
  - Vláknam patrí spoločný adresný priestor v rámci procesu
  - Hodnoty registrov medzi sebou nezdieľajú (každé vlákno má vlastné -> každé vlákno ma vlastný stack)

.exe?

- Ak má súbor príponu .exe, neznamená to, že je to spustiteľný súbor
- Ak má súbor príponu .mnau, neznamená to, že to nemôže byť spustiteľný súbor
- Súborová prípona je len informácia pre systém, aby vedel, čo spraviť s daným súborom
- Všetky dohody – HKEY_CLASSES_ROOT
- file.zri1, file.zri2, file.zri3, file.zri4

Portable Executable (PE)

- Porozumieť tomu, čo sa nachádza v súboroch nám pomáha lepšie pochopiť, ako funguje OS
- .exe, .dll, .sys, .com, ...
- Portable znamená, že formát je univerzálny naprieč platformou win32, bez ohľadu na verziu procesora.

Portable Executable

Zarovnania

- Zarovnanie sekcie – 00 _IMAGE_OPTIONAL_HEADER

```
.00427370:  00 00 00 00.00 00 00 00.00 00 00 00.00 00 00 00
.00427380:  00 00 00 00.00 00 00 00.00 00 00 00.00 00 00 00
.00427390:  00 00 00 00.00 00 00 00.00 00 00 00.00 00 00 00
.004273A0:  00 00 00 00.00 00 00 00.00 00 00 00.00 00 00 00
.004273B0:  00 00 00 00.00 00 00 00.00 00 00 00.00 00 00 00
.004273C0:  00 00 00 00.00 00 00 00.00 00 00 00.00 00 00 00
.004273D0:  00 00 00 00.00 00 00 00.00 00 00 00.00 00 00 00
.004273E0:  00 00 00 00.00 00 00 00.00 00 00 00.00 00 00 00
.004273F0:  00 00 00 00.00 00 00 00.00 00 00 00.00 00 00 00
.00428000:  00 10 00 00.8C 00 00 00.50 30 6C 30.11 31 3D 31   ▶ î   P0l0◀1=1
.00428010:  91 31 C8 31.F1 31 1D 32.8C 32 98 32.B9 32 CB 32  ‘1╚1±1↔2î2ś2╣2╦2
.00428020:  4D 33 69 33.BF 33 E4 33.F2 33 01 34.0B 34 1A 34  M3i3┐3ń3˙3⊕4♂4→4
.00428030:  24 34 52 34.64 34 82 34.A9 34 BB 34.C9 34 01 35  $4R4d4é4ę4┐4╔4⊕5
```

```
DWORD    SectionAlignment;    // In memory (0x1000)
DWORD    FileAlignment;       // In file (0x200)
```

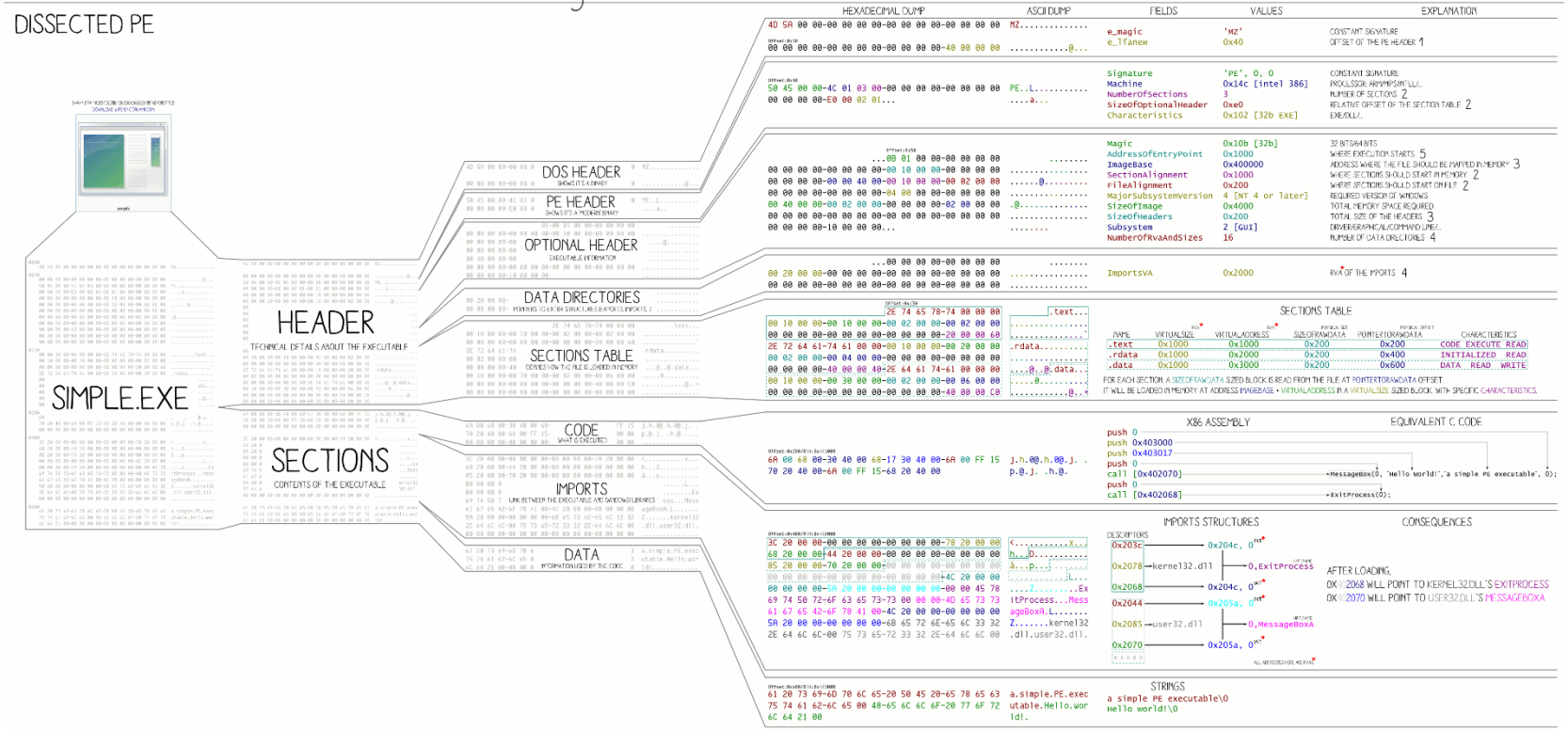
- Zarovnanie v kóde medzi funkciami – môže byť int 3 (CC), NOP (90), ...

```
012B1605 15 2B 01 64 15       adc         eax,1564012Bh
012B160A 2B 01                sub         eax,dword ptr [ecx]
012B160C 74 15                je          main+3h (012B1623h)
012B160E 2B 01                sub         eax,dword ptr [ecx]
012B1610 84 15 2B 01 CC CC    test        byte ptr ds:[0CCCC012Bh],dl
--- No source file -----------------------------------------------------
012B1616 CC                   int         3
012B1617 CC                   int         3
012B1618 CC                   int         3
012B1619 CC                   int         3
012B161A CC                   int         3
```

PE 101 — a windows executable walk-through

ANGE ALBERTINI
CORKAMI.COM

DISSECTED PE



THIS IS THE WHOLE FILE. HOWEVER, MOST PE FILES CONTAIN MORE ELEMENTS.
EXPLANATIONS ARE SIMPLIFIED, FOR CONCISENESS.

LOADING PROCESS

1 HEADERS

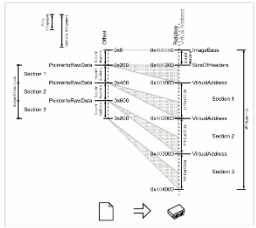
THE DOS HEADER IS PARSED
THE PE HEADER IS PARSED
IT'S OFFSET IS STORED AT OFFSET 0X3C IN THE DOS HEADER
THE OPTIONAL HEADER IS PARSED
IT FOLLOWS THE PE HEADER

2 SECTIONS TABLE

SECTIONS TABLE IS PARSED
IT IS LOCATED AT OFFSET (OPTIONALHEADER + SIZEOFOPTIONALHEADER)
IT CONTAINS NUMBEROFSECTIONS ELEMENTS
IT IS CHECKED FOR VALIDITY WITH ALIGNMENTS
FILEALIGNMENTS ARE SECTIONALIGNMENTS

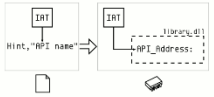
3 MAPPING

THE FILE IS MAPPED IN MEMORY ACCORDING TO
THE IMAGEBASE
THE SIZEOFHEADERS
THE SECTIONS TABLE



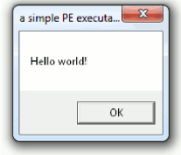
4 IMPORTS

DATADIRECTORIES ARE PARSED
THEY FOLLOW THE OPTIONALHEADER
THEIR NUMBER IS NUMBEROFRVAANDSIZES
IMPORTS ARE ALWAYS #2
IMPORTS ARE PARSED
EACH DESCRIPTOR SPECIFIES A DLLNAME
THIS DLL IS LOADED IN MEMORY
IAT AMOUNT ARE PARSED SIMULTANEOUSLY
FOR EACH API (INT)
IT'S ADDRESS IS WRITTEN IN THE IAT ENTRY



5 EXECUTION

CODE IS CALLED AT THE ENTRYPOINT
THE CALLS OF THE CODE GO VIA THE IAT TO THE APIS



NOTES

MZ HEADER AKA DOS_HEADER
STARTS WITH 'MZ' (INITIALS OF MARK ZBIKOWSKI MS-DOS DEVELOPER)

PE HEADER AKA IMAGE_FILE_HEADERS / COFF FILE HEADER
STARTS WITH 'PE\0\0' (PORTABLE EXECUTABLE)

OPTIONAL HEADER AKA IMAGE_OPTIONAL_HEADER
OPTIONAL ONLY FOR NON-STANDARD PES BUT REQUIRED FOR EXECUTABLES

RVA RELATIVE VIRTUAL ADDRESS
ADDRESS RELATIVE TO IMAGEBASE (AT IMAGEBASE, RVA = 0)
ALMOST ALL ADDRESSES OF THE HEADERS ARE RVAS
IN CODE, ADDRESSES ARE NOT RELATIVE

INT IMPORT NAME TABLE
NULL-TERMINATED LIST OF POINTERS TO HINT/NAME STRUCTURES

IAT IMPORT ADDRESS TABLE
NULL-TERMINATED LIST OF POINTERS
ON FILE IT IS A COPY OF THE INT
AFTER LOADING IT POINTS TO THE IMPORTED APIS

HINT
INDEX IN THE EXPORTS TABLE OF A DLL TO BE IMPORTED
NOT REQUIRED BUT PROVIDES A SPEED-UP BY REDUCING LOOK-UP

Data directory štruktúra

- Obsahuje umiestnenie dôležitých štruktúr spustiteľného súboru

Name	RVA	Size
Export	00224000	00005816
Import	00221000	0000226E
Resource	0022A000	000F0E00
Exception	00000000	00000000
Security	00000000	00000000
Fixups	0031B000	0001130C
Debug	00000000	00000000
Description	00000000	00000000
MIPS GP	00000000	00000000
TLS	00220000	00000018
Load config	00000000	00000000
Bound Import	00000000	00000000
Import Table	00000000	00000000
Delay Import	00000000	00000000
COM Runtime	00000000	00000000
(reserved)	00000000	00000000

Sekcie

- Slúžia na rozdelenie dát v spustiteľnom súbore podľa ich charakteristík:
- Spustiteľný kód, inicializované premenné, neinicializované premenné, importy, exporty, resource, relokácie a pod..
- Ich rozdelenie je z dôvodu „poriadku“ v súbore

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
00000438	00000440	00000444	00000448	0000044C	00000450	00000454	00000458	0000045A	0000045C
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	000F5000	00001000	000F4E00	00000600	00000000	00000000	0000	0000	60000020
.data	00129000	000F6000	00087A00	000F5400	00000000	00000000	0000	0000	C0000040
.tls	00001000	0021F000	00000200	0017CE00	00000000	00000000	0000	0000	C0000040
.rdata	00001000	00220000	00000200	0017D000	00000000	00000000	0000	0000	50000040
.idata	00003000	00221000	00002400	0017D200	00000000	00000000	0000	0000	40000040
.edata	00006000	00224000	00005A00	0017F600	00000000	00000000	0000	0000	40000040
.rsrc	000F1000	0022A000	000F0E00	00185000	00000000	00000000	0000	0000	40000040
.reloc	00012000	0031B000	00011400	00275E00	00000000	00000000	0000	0000	50000040
.bss	00000050	0032D000	00000000	00000000	00000000	00000000	0000	0000	C0000040

_IMAGE_SECTION_HEADER

- Name (.text, .data, .rdata, .reloc, .rsrc, .debug, .fero, ...)
- Názov sekcie nie je striktne stanovený, je to len „nepísané pravidlo“, avšak kernel (WinXP) obsahuje určité výnimky, pri ktorých overuje aj názov sekcie
- Attributes:

```
#define IMAGE_SCN_LNK_NRELOC_OVFL   0x01000000  // Section contains extended relocations.
#define IMAGE_SCN_MEM_DISCARDABLE   0x02000000  // Section can be discarded.
#define IMAGE_SCN_MEM_NOT_CACHED    0x04000000  // Section is not cachable.
#define IMAGE_SCN_MEM_NOT_PAGED     0x08000000  // Section is not pageable.
#define IMAGE_SCN_MEM_SHARED        0x10000000  // Section is shareable.
#define IMAGE_SCN_MEM_EXECUTE       0x20000000  // Section is executable.
#define IMAGE_SCN_MEM_READ          0x40000000  // Section is readable.
#define IMAGE_SCN_MEM_WRITE         0x80000000  // Section is writeable.
```

.text

- Obsahuje kód aplikácie
- Atribúty: Code | Readable | Executable
- Borland C++ používa namiesto .text názov sekcie CODE

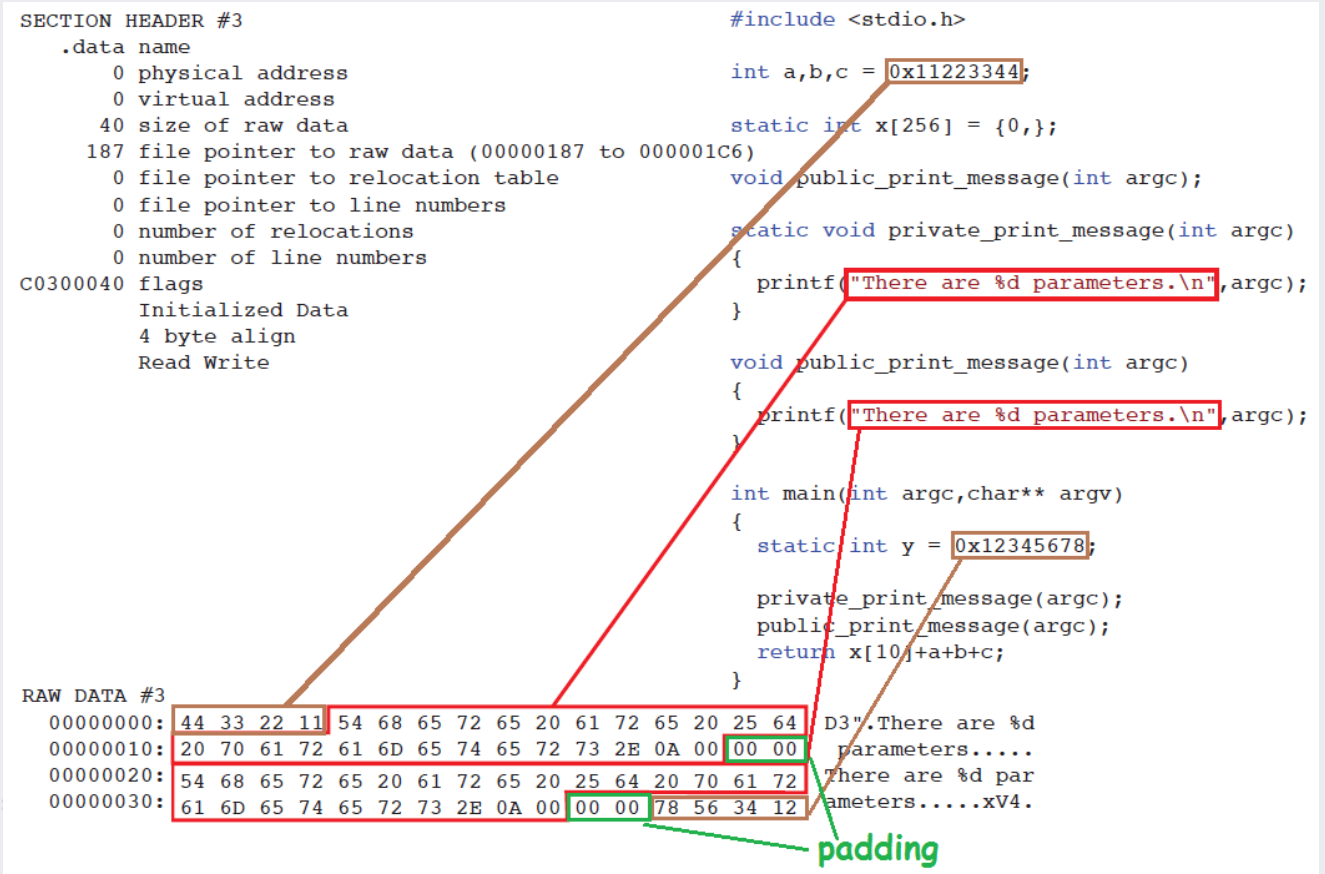
.bss (block started by symbol)

- Atribúty: Uninitialized
- Neinicializované statické a globálne premenné
- V PE súbore nezaberá nijaké miesto

```
#include <stdio.h>

int a,b,c = 0;                    Mapped into the .bss section

static int x[256] = {0,};

void public_print_message(int argc);

static void private_print_message(int argc)
{
  printf("There are %d parameters.\n",argc);
}

void public_print_message(int argc)
{
  printf("There are %d parameters.\n",argc);
}

int main(int argc,char** argv)
{
 static int y = 0;

  private_print_message(argc);
  public_print_message(argc);
  return x[10];
}
```

.data, .rdata

- Globálne a statické premenné, inicializované počas kompilácie



.reloc

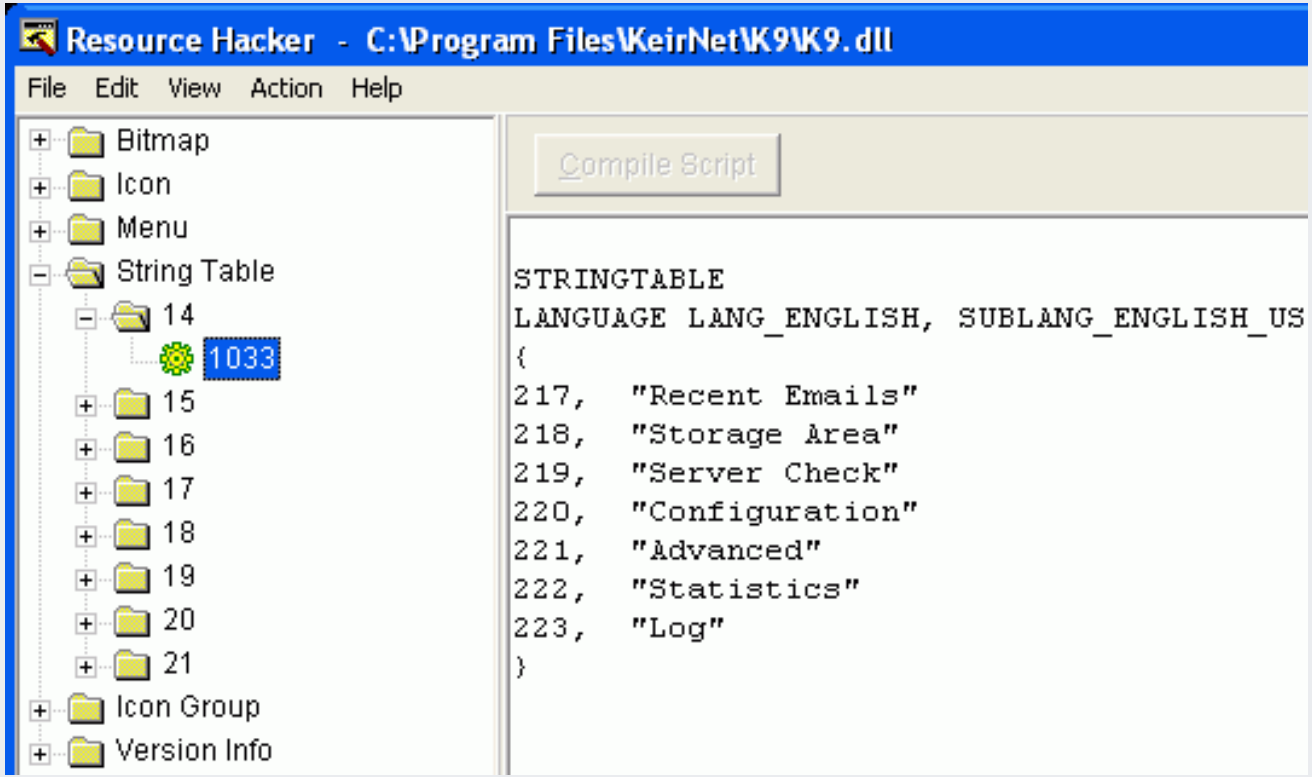
```
int gVar;
int main(int argc,char** argv) {
    int *ptr = &gVar;    // Assume &gVar = 0x402004
```

- Linker pred vytvorením PE súboru predpokladá, že sa pri spustení namapuje do konkrétneho adresného priestoru, napr. so začiatkom 0x400000 (Image base)
- Absolútne adresy počítajú s týmto začiatkom a preto môžu ďalej predpokladať, že na adrese 0x402004 sa bude napr. nachádzať globálna premenná „gVar“
- Ak však loader načíta súbor na adresu 0x700000 (ASLR, dll), musí spätne upraviť všetky absolútne adresy na túto premennú (&gVar = 0x702004)
- Mapovanie adries, ktoré je v takomto prípade zmeniť sa nachádza v sekcií .reloc
- Ak by loader namapoval súbor do predpokladaného adresného priestoru, celá sekcia .reloc by bola ignorovaná

.rsrc

Miesto na uloženie:

- Ikony
- Kurzory
- Položky menu
- Dialógové okná
- Stringy
- ...



Importy (.idata), export (.edata)

- Importy sú funkcie, ktoré sa nenachádzajú v moduli, ale sú ním volané. Takéto funkcie sa nachádzajú v externých knižniciach (.dll) a sú nimi exportované.
- Jedna knižnica môže importovať funkcie inej knižnice
- Tabuľka importov – pole deskriptorov ukončený nulovým deskriptorom
- sizeof(_IMAGE_IMPORT_DESCRIPTOR) = 0x14
- (size/0x14) – 1 = počet importovaných dll
- Aj .exe súbory môžu mať svoje exporty, podobne ako .dll

.tls

- Často používané aplikáciami s vláknami
- Inicializácia „globálnych“ premenných používaných vláknami
- Je možné registrovať „callback-y“, ktoré sa spúšťaj ešte pred EntryPointom

C and C++ [edit]

In C11, the keyword `_Thread_local` is used to define thread-local variables. The header `<threads.h>`, if supported, defines `thread_local` as a synonym for that keyword.

```
#include <threads.h>
thread_local int foo = 0;
```

Object Pascal [edit]

In Object Pascal (Delphi) or Free Pascal the *threadvar* reserved keyword can be used instead of 'var' to declare variables using the thread-local storage.

```
var
   mydata_process: integer;
threadvar
   mydata_threadlocal: integer;
```

https://github.com/corkami

Call Sleep(33120000)

Jmp cviko