



PE: specification vs. loader

Alexander Liskin

Senior Malware Analyst , Packer Analysis Team
Research & Development

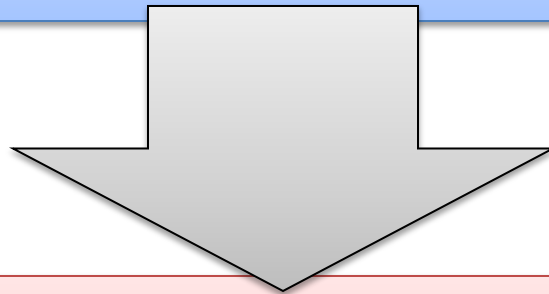
Cyprus, 2010

The background of the slide features a faint, stylized illustration. On the left, there is a silhouette of a ship with three masts and a large, rounded hull. On the right, there is a silhouette of a tractor with large, treaded wheels. The entire background is overlaid with a complex, light-colored grid pattern of intersecting lines.

Introduction

Specification and loader

Windows PE loader processes executable files not in full compliance with Microsoft PE & COFF specification



- It is possible to make PE files which
- 1) structure is at variance with specification
 - 2) run normally
 - 3) are not processed correctly by analysis tools

The background of the slide is a light gray illustration of a large, multi-decked sailing ship, possibly a galleon, with several masts and sails. The ship is positioned in the center and is partially obscured by a large, faint, white watermark that reads "Number of sections". The overall style is clean and modern, with a focus on the ship's structure.

Number of sections

Specific values

Specification runs as follows

2	2	NumberOfSections	The number of sections. This indicates the size of the section table, which immediately follows the headers.
---	---	------------------	--

It is presumed that every PE file has at least one section

Example of file with no sections

```
add [eax],al
add [eax],al
add [eax],al
```

Count of sections	0	Machine	Intel386
Symbol table	00000000[00000000]		Mon May 17 13:30:16 2010
Size of optional header	01E0	Magic optional header	010B
Linker version	9.00	OS version	5.00
Image version	0.00	Subsystem version	5.00
Entry point	00000218	Size of code	000000E0
Size of init data	00000000	Size of uninit data	00000000
Size of image	000002D0	Size of header	000001F0
Base of code	000001F0	Base of data	000002D0
Image base	00400000	Subsystem	GUI
Section alignment	00000004	File alignment	00000004
Stack	00100000/00001000	Heap	00100000/00001000
Checksum	00000000	Number of dirs	16

```
insb
xor esi,[edx]
insb
```

Example of file with no sections

```
add     [eax],al
add     [eax],al
add     [eax],al

Number  Name      VirtSize  RVA      PhysSize  Offset  Flag
Cursor's out of sections

insb
xor     esi,[edx]
insb
```

This file runs normally on



- Hiew v8.03 (November 2009) does not recognize this file as PE
- Hiew v8.10 (February 2010) handles it correctly

```
===== at 00000000 =====  
Signature                5A4D  
Bytes on last page      0090  
Pages in file           0003  
Relocations count       0000  
Paragraphs in header    0004  
Minimum memory          0000  
Maximum memory          FFFF  
SS:SP setting           0000:00B8  
Checksum                0000  
CS:IP setting           0000:0000  
Relocations table address 0040  
Overlay number          0000  
Overlay length          FFFF  
NewExe offset           000000C8  
Entry point             00000040
```

WORD NumberOfSections;

```
000000C0: 00 00 00 00-00 00 00 00-50 45 00 00-4C 01 00 00 PE L@  
000000D0: A8 0C F1 4B-00 00 00 00-00 00 00 00-E0 01 03 01  н♀èK p@#@  
000000E0: 0B 01 09 00-E0 00 00 00-00 00 00 00-00 00 00 00  δ@0 p  
000000F0: 18 02 00 00-F0 01 00 00-D0 02 00 00-00 00 40 00  t@ E@ u@ e
```

Specification runs as follows

At the beginning of an object file, or immediately after the signature of an image file, is a standard COFF file header in the following format. Note that **the Windows loader limits the number of sections to 96.**

Actual maximal number of sections is

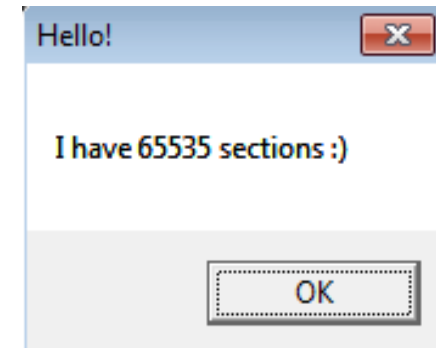
95 on



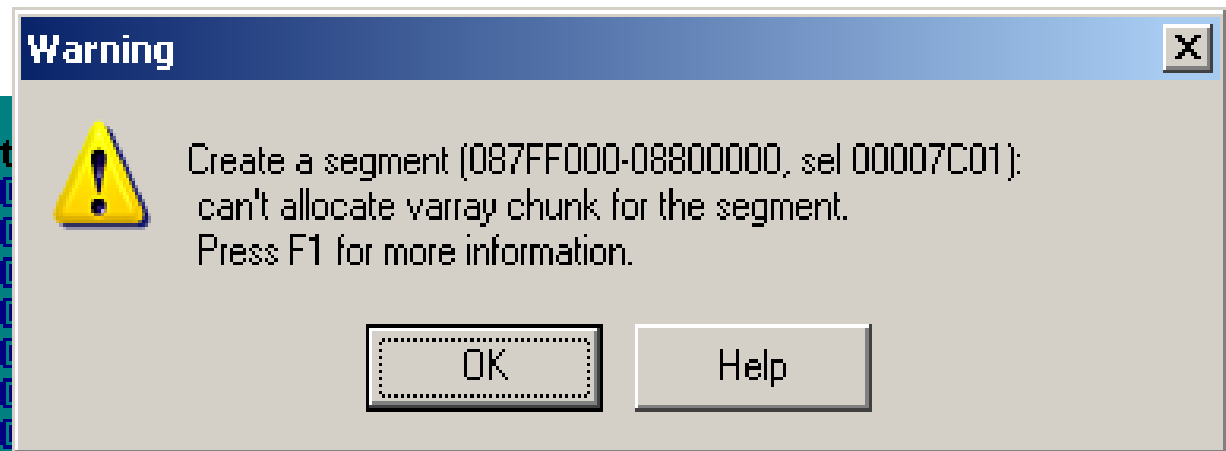
65535 on



and



- IDA v5.6 fails to load file with 65535 sections (by default)
- Hiew v8.10 needs 20 seconds to open it



Number	Name	Virt				
65524	sectFFF4	0000				
65525	sectFFF5	0000				
65526	sectFFF6	0000				
65527	sectFFF7	0000				
65528	sectFFF8	0000				
65529	sectFFF9	0000				
65530	sectFFFA	0000				
65531	sectFFFB	00001000	107F9000	00000200	027FF200	E0000020
65532	sectFFFC	00001000	107FA000	00000200	027FF400	E0000020
65533	sectFFFD	00001000	107FB000	00000200	027FF600	E0000020
65534	sectFFFE	00001000	107FC000	00000200	027FF800	E0000020
65535	sectFFFF	00001000	107FD000	00000200	027FFA00	E0000020

The background of the slide features a stylized, light gray illustration of a large sailing ship with multiple masts and sails, set against a backdrop of abstract, flowing lines and a grid pattern. The ship is positioned in the middle ground, appearing to move across the water.

Section header

Anomalous offsets and sizes

Specification runs as follows

20	4	PointerToRawData	<p>The file pointer to the first page of the section within the COFF file. For executable images, this must be a multiple of FileAlignment from the optional header. For object files, the value should be aligned on a 4-byte boundary for best performance. When a section contains only uninitialized data, this field should be zero.</p>
----	---	------------------	--

Section header: unaligned physical offset

Section headers of files packed by Upack are at variance with specification

```
dec     esp
imul   esp,[edx][072],041797261 ;'Ayra'
add     [eax],al
```

Count of sections	3	Machine	Intel386
Symbol table	FF50AD00[7CEB3476]		Sat Jan 24 02:39:42 2004
Size of optional header	0148	Magic optional header	010B
Linker version	76.11	OS version	4.00
Image version	0.57	Subsystem version	4.00
Entry point	00001018	Size of code	694C6461
Size of init data	72617262	Size of uninit data	00004179
Size of image	0001F000	Size of header	00000200
Base of code	00000010	Base of data	0000B000
Image base	00400000	Subsystem	GUI
Section alignment	00001000	File alignment	00000200
Stack	00100000/00001000	Heap	00100000/00001000
Checksum	00000000	Number of dirs	10

```
add     [eax],al
add     al,dh
add     [eax],eax
```

Section header: unaligned physical offset

Section headers of files packed by Upack are at variance with specification

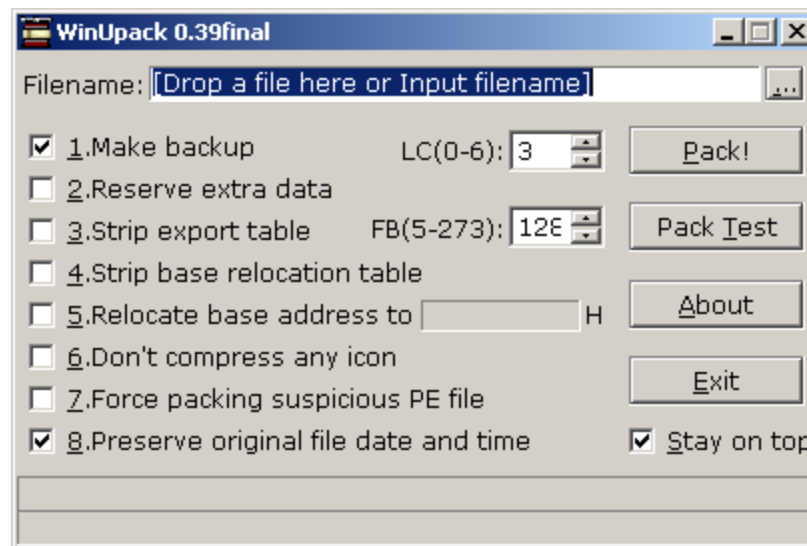
```
dec     esp
imul   esp,[edx][072],041797261 ;'Ayra'
add    [eax],al

Number  Name      VirtSize  RVA      PhysSize  Offset  Flag
-----
1 PS     Флывч | 00012000 00001000 000001F0 00000010 E0000060
2                0000B000 00013000 00006908 00000200 E0000060
3 0xA                00001000 0001E000 000001F0 00000010 E0000060

add    [eax],al
add    al,dh
add    [eax],eax
```

Section header: unaligned physical offset

But they run normally on



Section header: unaligned physical offset

PE loader calculates section data offset like this

```
// size of disk sector
#define SECTOR_SIZE 512
// ...
// size of memory page
#define PAGE_SIZE 4096
// ...
// align down
#define ALIGN_DOWN(X, Y) ((X) & (~((Y) - 1)))
// ...
IMAGE_SECTION_HEADER SectHdr;
IMAGE_NT_HEADERS NtHdr;
long SectOffs;
// ...
// calculate offset to read section from
SectOffs = (NtHdr.OptionalHeader.SectionAlignment >= PAGE_SIZE) ?
    ALIGN_DOWN(SectHdr.PointerToRawData, SECTOR_SIZE) :
    SectHdr.PointerToRawData;
```

Section header: large physical size

Specification runs as follows

16	4	SizeOfRawData	The size of the section (for object files) or the size of the initialized data on disk (for image files) . For executable images, this must be a multiple of FileAlignment from the optional header. If this is less than VirtualSize, the remainder of the section is zero-filled. Because the SizeOfRawData field is rounded but the VirtualSize field is not, it is possible for SizeOfRawData to be greater than VirtualSize as well. When a section contains only uninitialized data, this field should be zero.
----	---	---------------	---

Section header: large physical size

Section physical size could be much bigger
then file size

```
add    [eax],al
add    [eax],al
add    [eax],al
```

Number	Name	UirtSize	RVA	PhysSize	Offset	Flag
1	.text	00000026	00001000	00000200	00000400	60000020
2	.rdata	000000AA	00002000	7FFFFFFF	00000600	40000040
3	.rsrc	000003A0	00003000	00000400	00000800	40000040

```
add    [eax],al
add    [eax],al
add    [eax],al
```

Section header: unaligned physical offset

But such files run normally on



Section header: large physical size

Microsoft installer's resource section case

```
inc     ecx
push   ecx
call   .001005977 --↑B
```

Number	Name	VirtSize	RVA	PhysSize	Offset	Flag
1	.text	000078A0	00002000	00007A00	00000400	60000020
2	.data	000110D4	0000A000	00000200	00007E00	C0000040
3	.rsrc	0000144C	0001C000	002B2400	00000800	40000040

```
jmps   .0010059EE --↓H
jtest  cl,cl
jz     .001005A07 --↓F
```

Specification runs as follows

8	4	VirtualSize	The total size of the section when loaded into memory. If this value is greater than SizeOfRawData, the section is zero-padded. This field is valid only for executable images and should be set to zero for object files.
---	---	-------------	---

Section header: large physical size

Virtual size of some nonempty sections of files built by Watcom C/C++ is zero

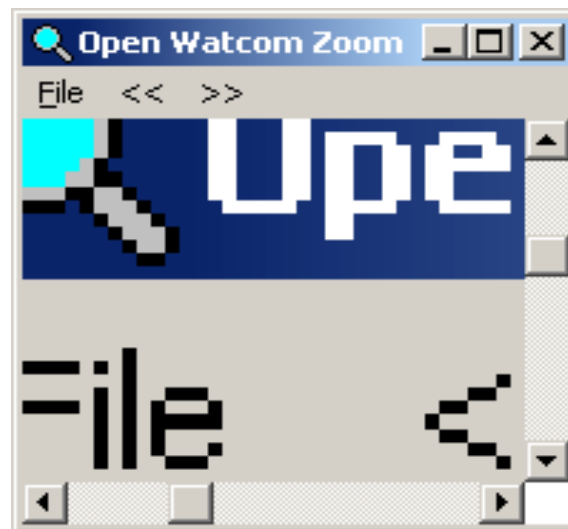
```
jns      .000403072 --↓6
popad
jnc      .000403078 --↓7
```

Number	Name	VirtSize	RVA	PhysSize	Offset	Flag
1	AUTO	000079C6	00001000	00007A00	00000400	60000020
2	.idata	00000ADB	00009000	00000C00	00007E00	C0000040
3	DGROUP	00001C60	0000A000	00000400	00008A00	C0000040
4	.edata	00000000	0000C000	00000200	00008E00	40000040
5	.reloc	00000000	0000D000	00000800	00009000	42000040
6	.rsrc	00000000	0000E000	00001400	00009800	40000040

```
add     [ecx],eax
add     eax,[ebx]
add     eax,[ebx]
```

Section header: unaligned physical offset

But such files run normally on



The background of the slide features a faint, stylized illustration of a large sailing ship with multiple masts and sails, set against a backdrop of abstract, flowing lines and a light gray gradient. The ship is positioned in the middle ground, appearing to move from left to right.

Conclusion

Where we come to

- Mismatch of PE specification and PE loader gives possibilities of creating executables which deceive analytics and analysis tools
- File may not be corrupted even if it is incorrectly processed by analysis tools, you need to run it to figure out



Thank you!
Questions?

Alexander Liskin

Packer Analysis Team