doT.Net

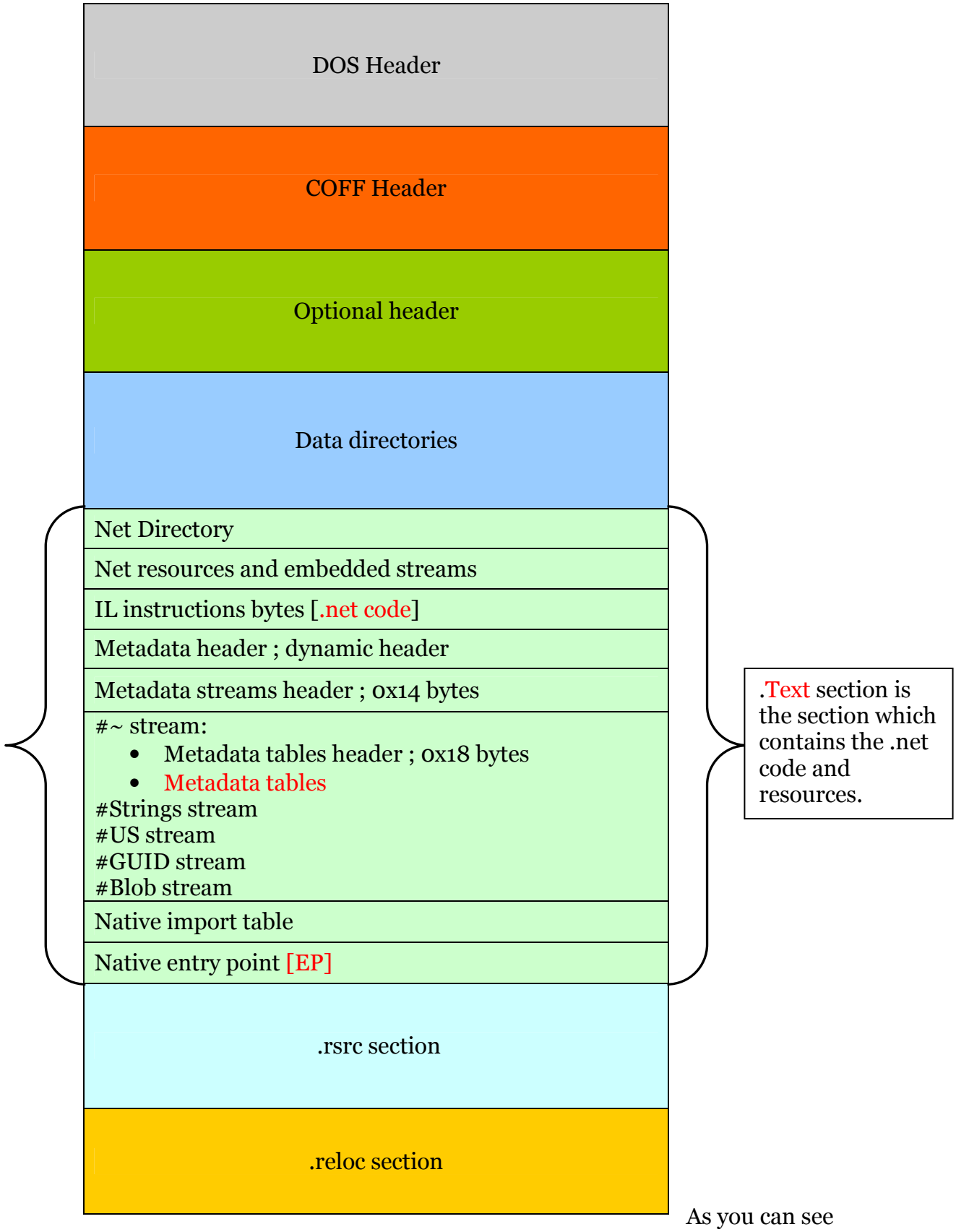
Resources

Welcome back, this is not a cracking tutor, it's rather a paper I want to share with you and my friend UFO about the managed resources in .net executables, I got the idea when UFO was trying to improve his {SmartKiller} to make it extract certain resources from an assembly without need for manual dumping.

First of all I will assume that you have a simple knowledge of what we are talking about! Or at least you have read some previous tutors on PE files; this is not the place to explain all the ins and outs of PE files.

I recommend that everyone reads a paper by **Ntoskrnl** which discusses in detail the ins and outs of the .net PE file, I will post it with this paper to give you more info on the entire structure, and you should read his paper on Manifest streams too before this to get a better idea.

A standard .net executable is nothing but a standard PE file, but how does the .net stuff get arranged in the file anyway? What we are trying to focus on in the executable is the part that contains the .net stuff, I mean the .text section of course, this is where the compiler puts all the .net stuff finally, I will try to give you a better idea about this shit now, look at this table below which illustrates the structure of a simple .net executable.



As you can see there's nothing new here, it's just a typical executable file, but we will discuss the part that contains the .net stuff, I mean the .text section.

- dot Net resources and embedded streams

A resource is binary data that you can add to the executable file of a Windows-based application. A resource can be either standard or defined. The data in a standard resource describes an icon, cursor, menu, dialog box, bitmap, enhanced metafile, font, accelerator table, message-table entry, string-table entry, or version information. An application-defined resource, also called a custom resource, contains any data required by a specific application.

Check this dump for CrackME #9, this assembly contains a music file named "doomie" which was saved in the final assembly as an embedded resource stream during the final compilation process.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000FB0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000FC0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000FD0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000FE0	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	20	2E	11	00	00	00	00	00	48	00	00	00	02	00	05	00	.....H.....
00001010	D0	E8	10	00	20	45	00	00	01	00	00	00	4E	00	00	06	Ðè.. E.....N...
00001020	50	20	00	00	9B	B7	10	00	00	00	00	00	00	00	00	00	P ..›·.....
00001030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00001040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00001050	65	3A	0D	00	49	4D	50	4D	30	35	20	2D	20	44	6F	6F	e:..IMPM05 - Doo
00001060	6D	69	65	00	00	00	00	00	00	00	00	00	00	00	00	00	mie.....
00001070	00	00	04	10	1A	00	00	00	1B	00	1F	00	17	02	00	02	.....
00001080	09	00	07	00	80	30	03	7D	80	00	1B	00	C4	01	00	00	....€0.}€...ĵ...
00001090	00	00	00	00	20	20	20	20	20	20	20	20	20	20	20	20	....
000010A0	20	20	20	20	20	20	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	□□□□□□□□□□
000010B0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	□□□□□□□□□□□□□□□□
000010C0	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	□□□□□□□□□□□□□□□□
000010D0	FF	FF	FF	FF	40	40	40	40	40	40	40	40	40	40	40	40	□□□□@@@@@@@@@@@@
000010E0	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	@@@@@@@@@@@@@@@@

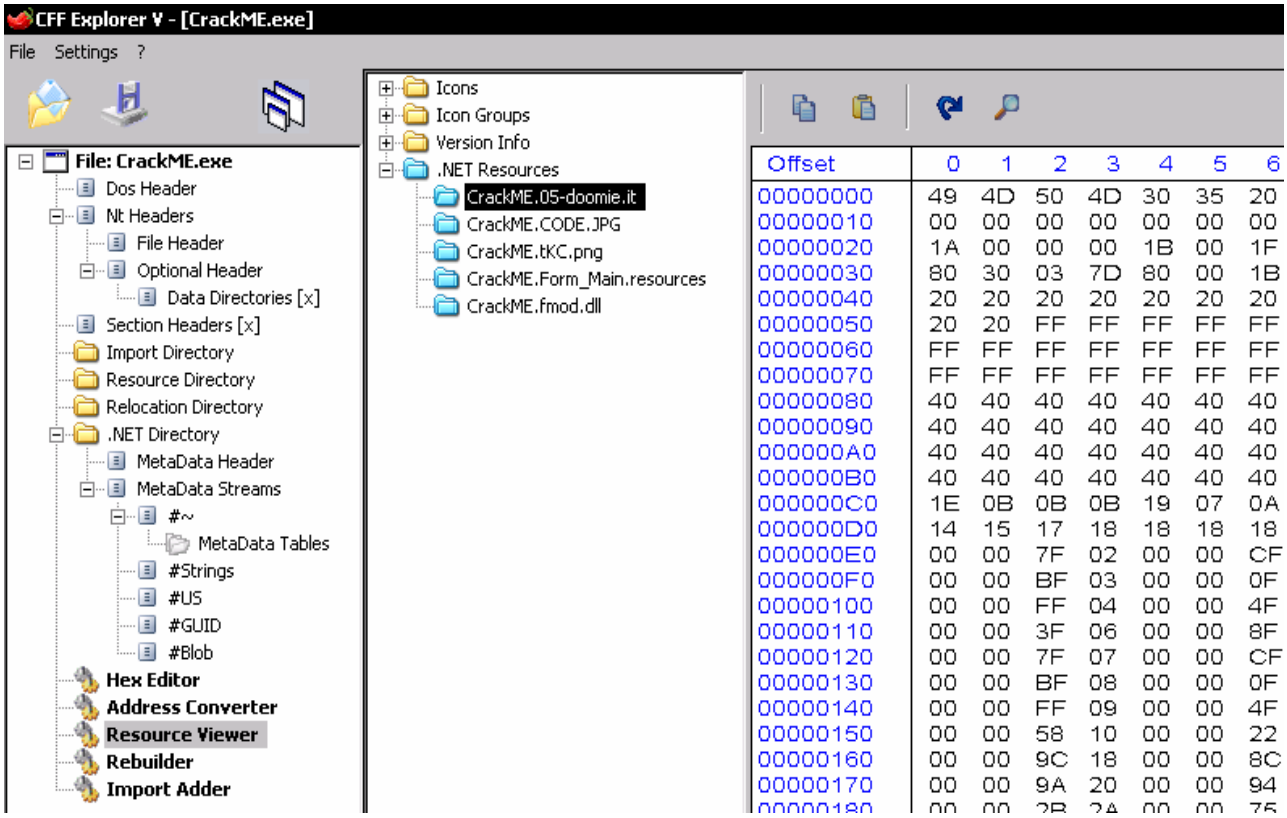
The .text section in this PE file start at offset 0x0001000 and you can see the bytes of the embedded stream start at offset 0x0001050.

The first byte of the stream here begins at offset 0x0001054 right after byte 0x49, the first 4 bytes before this byte represent a 32 bit integer which is the raw size of this stream and it's 866917 bytes in length here, so what will we find after 866917 bytes from this offset ?
If the assembly has more than one embedded resource stream then you will find the next resource.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
000D4A00	F5	F5	F5	F5	F5	F5	F5	F5	F5	F5	F5	F5	F5	F5	F5	F3	õ
000D4A10	F3	F3	EE	EE	EE	EE	EE	EE	EE	EE	EE	EE	EB	EB	E7	E7	óóîîîîîîîîëëçç
000D4A20	E7	E4	E4	E4	E4	DF	DF	DD	DD	DD	DD	DF	E4	E7	EB	EB	çääääßßÝÝÝßäçëë
000D4A30	EB	EB	E7	E7	E4	E4	E4	DF	DF	DF	E4	E4	E7	E7	E7	EB	ëëççäääßßßäçççë
000D4A40	EB	EB	EB	EB	EB	E7	E7	E4	DF	DF	DF	E4	E7	EB	EB	EE	ëëëëëçççäßßßäçëëî
000D4A50	F3	F3	EE	EE	EB	EB	E7	E7	E7	EB	EB	EE	F3	F5	F5	F5	óóîîëëçççëëîôõõõ
000D4A60	F5	F5	F5	F5	F5	F5	F5	F5	F9	F9	00	00	00	00	00	00	õõùù......
000D4A70	01	01	06	09	0D	0F	0F	0D	09	09	09	09	09	0D	0D	0D	.....
000D4A80	0D	0D	09	09	09	09	09	0D	0D	0F	14	17	17	17	14	0F	.....
000D4A90	0F	0D	0D	0D	0D	0F	0F	0F	0F	14	14	14	14	14	0F	0F	.....
000D4AA0	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0F	0D	.....
000D4AB0	0D	09	01	00	F9	F9	F9	00	01	0A	08	00	00	FF	D8	FF	....ùùù..ÿØÿ
000D4AC0	E0	00	10	4A	46	49	46	00	01	01	01	00	60	00	60	00	à..JFIF.....`.`
000D4AD0	00	FF	E1	00	16	45	78	69	66	00	00	49	49	2A	00	08	.ÿá..Exif..II*..
000D4AE0	00	00	00	00	00	00	00	00	00	FF	DB	00	43	00	08	06	.....ÿÛ.C...
000D4AF0	06	07	06	05	08	07	07	07	09	09	08	0A	0C	14	0D	0C	.....

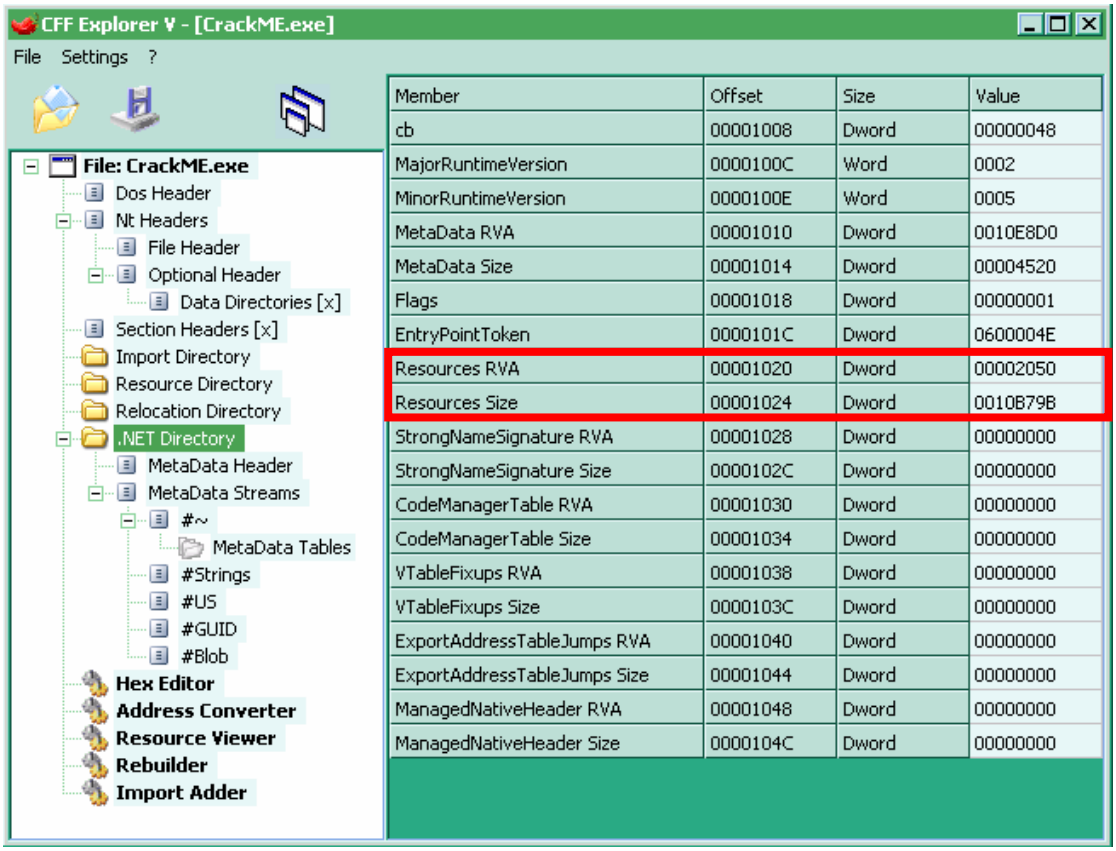
Here we see the end of the previous resource stream and the beginning of the next resource stream which starts from the offset 0x000D4AB9 and as we said before the first 4 bytes represent the size of this resource but the actual bytes of this stream which is a JPG image start at offset 0x000D4ABD.

You can use CFF explorer to view the managed resources in the assembly.



One last point I want to mention here which is how to find the file offset for a particular resource stream ?

Although it's early here to answer this question because the answer depends on the CLI header of the assembly, and I mean the "Resources RVA" field in that structure which leads us to the first byte of the .net resources section, The resources streams are written to the final assembly file according to the order they are listed in the "ManifestResources" table in meta data tables, Notice this.



Member	Offset	Size	Value
cb	00001008	Dword	00000048
MajorRuntimeVersion	0000100C	Word	0002
MinorRuntimeVersion	0000100E	Word	0005
MetaData RVA	00001010	Dword	0010E8D0
MetaData Size	00001014	Dword	00004520
Flags	00001018	Dword	00000001
EntryPointToken	0000101C	Dword	0600004E
Resources RVA	00001020	Dword	00002050
Resources Size	00001024	Dword	0010B79B
StrongNameSignature RVA	00001028	Dword	00000000
StrongNameSignature Size	0000102C	Dword	00000000
CodeManagerTable RVA	00001030	Dword	00000000
CodeManagerTable Size	00001034	Dword	00000000
VTableFixups RVA	00001038	Dword	00000000
VTableFixups Size	0000103C	Dword	00000000
ExportAddressTableJumps RVA	00001040	Dword	00000000
ExportAddressTableJumps Size	00001044	Dword	00000000
ManagedNativeHeader RVA	00001048	Dword	00000000
ManagedNativeHeader Size	0000104C	Dword	00000000

Now the "Resources RVA" is an RVA value so use the Address converter to get the file offset and then you will get the offset for the first byte of the first managed resource in the assembly, but as we mentioned before the first 4 bytes are the size of the managed resource!

Now the number of managed resources can be retrieved by reading the entire resources block and separating the entire block according to the 4 byte size indicator at the beginning of every single managed resource stream in the entire block until the reading pointer reaches the end of the resources block, another method to do this is by reading the number of rows in the "ManifestResources" table in meta data tables.

Kurapica
Saturday, May 26, 2007
gREETz.