	Adding sections to PE Files Enhancing functionality of programs by adding extra code	papers Papers
10 July 1999	by c0v3rt+	
	Courtesy of Reverser's page of reverse engineering	
fra_00xx 98xxxx c0v3rt+ 1100 PA PC	*Header galore! A very good essay by c0v3rt+* *Reversers and protectors are well advised to read, re-read and put in practice the approaches, tricks and knowledges listed in this paper, like the ways to change the virtual size of an exe section.* *It would also be extremely interesting to add some other essays, or papers, about enhancing functionality of programs by adding extra code, since this is truly one of the most important "threads" in our trade. (I hope that c0v3rt+ will send more of them himself :-)* *Protectors should carefully study the following text.*	
	There is a crack, a crack in everything That's how the light gets in	
Rating	**(*)Beginner (*)Intermediate ( )Advanced ( )Expert**	

How to add new sections to a PE file, useful to anyone wanting to add functionality to a program and (in some cases) to crack certain protection schemes.

Adding sections to PE Files
Enhancing functionality of programs by adding extra code

Written by c0v3rt+

Introduction

It is often necessary to add code to a program in order to either crack a protection scheme or more usually to add functionality to it. To do this an understanding of the PE file format is required. In this article as well as learning how to add a new section to a PE file you will also learn a little more about the structures in these files.

Tools required

dumpbin.exe (comes with Windows 95/98)
fred.exe (or your favorite hex editor)
MS VC++ (optional)

Essay

In this article I will be making reference to the following sources of information

A description of the PE file format and the various structures and sections can be found on the Microsoft site at

http://msdn.microsoft.com/library/specs/msdn_pecoff.htm

If you are using the Microsoft Visual C++ Compiler you will also find some useful definitions for the structures you will find in a PE file in the header file **winnt.h**

Very briefly, a PE file consists of a DOS header (defined as IMAGE_DOS_HEADER in winnt.h) followed by a NT header (IMAGE_NT_HEADERS). Then there are a number of section headers which define each of the sections in the file. The sections themselves then follow. Each PE section has various characteristics (as defined in the section headers) which define whether the section is Read only, Read/Write whether the section is code, initialized data, uninitialized data etc. Further information on this can be found in the Microsoft documentation

If you use the dumpbin.exe tool on some of your favorite executable files you will be able to see what sections they contain. For example the following command

dumpbin pbrush.exe

will give the following output

```
Microsoft (R) COFF Binary File Dumper Version 3.10.6038
Copyright (C) Microsoft Corp 1992-1996. All rights reserved.


Dump of file pbrush.exe

File Type: EXECUTABLE IMAGE

     Summary

        1000 .idata
        1000 .reloc
        1000 .rsrc
        1000 .text
```

The main section that interests us for now is the .text section. This is simply the place where the program code resides. You can find out what the other sections are in the Microsoft documentation.

To see more detail you can use the command

dumpbin /headers pbrush.exe

This will give you information about the DOS and the NT headers and separate information about each section in the file. If you want to find out how else dumpbin can be used just enter the command with no parameters

To crack most programs you can get by with only a moderate knowledge of the PE format. If you want to start adding large amounts of code then you will need to know more. I will now proceed to describe several methods you can use to add code to an existing executable.

Adding to an existing section, technique 1

Lets look at pbrush again. If you look at the sections using dumpbin you will see some information on the .text section part of which is shown below

```
SECTION HEADER #1
   .text name
      AB virtual size
    1000 virtual address
    1000 size of raw data
    1000 file pointer to raw data
```

The *virtual size* represents the amount of actual code. The *size of raw data* defines the amount of space taken up in the file sitting on your hard disk. Note that the virtual size in this case is lower than that on the hard disk. This is because compilers often have to round up the size to align a section on some boundary, in this case it rounds up to the next 0x1000 boundary.

Now pbrush.exe is unusual in that it does so little, only 0xAB bytes of code! In fact all that pbrush does is to spawn mspaint.exe which is why there is so little code.

What is of interest here is that the virtual size is less than the size of the raw data. Can we use that extra space? The answer is yes.

If you were to dis-assemble pbrush.exe using IDA-Pro then you would find that there appears to be a dis-continuity between address 0x4010ab (base address 0x400000 + virtual address 0xab) and address 0x402000. Use your favorite hex editor (fred.exe) to look at the contents of the pbrush.exe file at offset 0x10ab (file pointer to raw data 0x1000 plus virtual size 0xAB) and you will find that the file is full of 0x00. This would appear to be an ideal location to place your own code.

pbrush.exe is unusual in that it has such a large amount of space free at the end of the .text section. You will find anywhere from zero to 0xfff bytes free at the end of any section. This is because each section has to be aligned on a 0x1000 byte boundary. The only other thing you will need to do other than adding your code to the 0x0f55 free bytes at the end of this section is to change the virtual size of the section. The easiest way to find the section header in the exe file is to search for the text string ".text" and then look for the value 0xAB following it. In this case the virtual size can be found at file offset 0x180 as you can see from the following hex dump

```
 000170   00 00 00 00 00 00 00 00-2E 74 65 78 74 00 00 00 ï¿½ï¿½ .........text...
 000180   AB 00 00 00 00 10 00 00-00 10 00 00 00 10 00 00 ï¿½ï¿½ ................
 000190   00 00 00 00 00 00 00 00-00 00 00 00 20 00 00 60 ï¿½ï¿½ ............ ..`
```

The virtual size is a 32 bit number so to change it to the full amount of space you have available (0x1000) you should make the following change

```
 000170   00 00 00 00 00 00 00 00-2E 74 65 78 74 00 00 00 ï¿½ï¿½ .........text...
 000180   00 10 00 00 00 10 00 00-00 10 00 00 00 10 00 00 ï¿½ï¿½ ................
 000190   00 00 00 00 00 00 00 00-00 00 00 00 20 00 00 60 ï¿½ï¿½ ............ ..`
```

One other point. Each section has characteristics defined for it. Usually text sections are defined to be code, read-only and execute. This would mean that if you were to try and put both code and data into the section you would get a page fault. To correct this you need to change the characteristics. In this case the characteristics are 0x60000020 (starting at file offset 0x19c). Looking at the winnt.h file you can find it is made up of the following characteristics

```
#define IMAGE_SCN_CNT_CODE                     0x00000020  // Section contains code.
#define IMAGE_SCN_MEM_EXECUTE                  0x20000000  // Section is executable.
#define IMAGE_SCN_MEM_READ                     0x40000000  // Section is readable.
```

The full characteristics are defined by the boolean OR of all these flags. This section is Code, Execute, Readable. To write to it you need to add the flag

```
#define IMAGE_SCN_MEM_WRITE                    0x80000000  // Section is writeable.
```

This gives you the new value 0xE00000020 which when patched into the executable gives you the following dump

```
 000170   00 00 00 00 00 00 00 00-2E 74 65 78 74 00 00 00 ï¿½ï¿½ .........text...
 000180   00 10 00 00 00 10 00 00-00 10 00 00 00 10 00 00 ï¿½ï¿½ ................
 000190   00 00 00 00 00 00 00 00-00 00 00 00 20 00 00 E0 ï¿½ï¿½ ............ ..`
```

If you now dumpbin the pbrush.exe file with the /headers flag you should see that the .text section virtual size is not 0x1000 and that the section is marked *Code, Execute, Read, Write* as we require. There is now no problem adding your new code and data to the end of the .text section

Of course there may well be times when the amount of free space is insufficient for your purposes. In this case you will need to take an alternative approach. which will be described in the following method

Adding to an existing section, technique 2

If there is not sufficient space at the end of the text section you will need to extend it. This poses a number of problems.

1. If the section is followed by other sections then you will need to move the following sections up to make room

2. There are various references within the file headers that will need to be adjusted if you change the file size.

3. References between various sections (such as references to data values from the code section) will all need to be adjusted. This is practically impossible to do without re-compiling and re-linking the original file.

Most of these problems can be avoided by appending to the last section in the exe file. This however poses a number of further problems, none of which are insurmountable.

If you dumpbin a selection of PE files you will see that they often have the .reloc section as the last section. Take care that you actually used dumpbin with the /headers flag since although the summary shows the sections it does not put them in the correct order! Looking at the example program pbrush.exe we can see that the last section has the following characteristics

```
SECTION HEADER #4
  .reloc name
      34 virtual size
    4000 virtual address
    1000 size of raw data
    4000 file pointer to raw data
       0 file pointer to relocation table
       0 file pointer to line numbers
       0 number of relocations
       0 number of line numbers
42000040 flags
         Initialized Data
         Discardable
         (no align specified)
         Read Only
```

The size of this section in the exe file is 0x1000 but only the first 0x34 bytes are actually being used. We can add 0xFCC bytes to this section. Let us say however that we want to enhance the functionality of pbrush.exe by adding more than this amount of code and data. We can do this since there are no sections following the .reloc section to get in the way. We just extend the size of the exe file.

We will however need to modify the characteristics of this section by changing the flags. We should add the IMAGE_SCN_CNT_CODE, the IMAGE_SCN_MEM_EXECUTE and the IMAGE_SCN_MEM_WRITE flags and we should remove the IMAGE_SCN_MEM_DISCARDABLE flag in order to ensure that the page remains in memory. This will give us the value of 0xE0000060

We will of course need to change the *virtual size* and the *size of raw data* in the section header to be in line with the amount of code/data we have now added.

There are however a couple of values in the NT headers that will need to be adjusted accordingly. Looking at the dumpbin once again with flag /headers you can see the following information

```
OPTIONAL HEADER VALUES
     10B magic #
    3.10 linker version
     200 size of code
     C00 size of initialized data
       0 size of uninitialized data
    100C address of entry point
    1000 base of code
    2000 base of data
         ----- new -----
  400000 image base
    1000 section alignment
    1000 file alignment
       2 subsystem (Windows GUI)
    4.00 operating system version
    0.00 image version
    4.00 subsystem version
    5000 size of image
     400 size of headers
    A9C1 checksum
  100000 size of stack reserve
    1000 size of stack commit
  100000 size of heap reserve
    1000 size of heap commit
```

The critical value is *size of image* If you try to just change the size of the last section without changing this value as well then you will get an error from Windows to the effect that there is not sufficient memory to run the program. To avoid this increase this value in the NT headers by a similar amount. The *size of code* and the *size of initialized data* seem to be less critical but as a matter of course I increase these by a similar amount.

To determine the file offset for these values you will need to look at the definitions for NT headers. Look for IMAGE_NT_HEADERS and IMAGE_DOS_HEADER in the file winnt.h or work it out from the Microsoft documentation.

Adding a new section

The ability to extend the size of the last section of a PE file will normally be all you need. I have however come across a rather widely used CD-ROM protection scheme that can be more easily cracked by adding new sections to the end of the file. The reason why this is possible is due to the way that the code self-checks against patches. A brief outline of this protection is given below

The program has .txt and .txt2 sections in addition to the normal .text section and each of these contains code. If you try to dis-assemble the program however you will find that section .txt is encrypted. The .text section contains the startup code and the standard Visual C++ library routines, the .txt2 section contains much of the code to do SoftIce checks and the decryption routines for the .txt section.

If you attempt to patch the program, for example to remove the SoftIce checks you will find that a self-checking program will cause the program to crash. On further investigation it becomes apparent that they have not implemented a simple sum-check with a comparison against the expected value. This would be easy to patch out as well. They actually use the sum-check as the decryption key to unlock the .txt section. Thus a simple patch in either the .text section or in the .txt2 section will cause the decryption to fail and the program to crash when it tries to execute the decrypted code.

The **BIG** mistake they have made however is that in order to calculate the sum-check on the .txt2 or .text sections they determine the address of these sections from the PE file headers rather than the absolute address of the section. You can fool the program into checking an un-modified copy of the section by making an exact copy of it at the end of the file and with the original section name. The original section is renamed. e.g .txt2 is copied to the end of the file and it takes the name .txt2, the original section is renamed to .txt2b You can now make patches to the original (and still executing) code in the .txt2b section and the self-checking will not detect it.

In order to crack this program I created a utility that automated the section copy process. The outline of the process is as follows.

First scan the section headers to locate the *file pointer to raw data* and the *size of raw data* for the section you want to copy. Read the whole section from the exe file.

Append the contents of the section to the end of the file

You will now need to make some changes to the header files, you need to change the *size of code, size of initialized data* and the *size of image* as described above. Increase these by the *virtual size* given in the section. You also need to add a new section header. To do this locate the section header for the original section. A section header is defined as follows (taken from the winnt.h file)

```
typedef struct _IMAGE_SECTION_HEADER {
    BYTE    Name[IMAGE_SIZEOF_SHORT_NAME];
    union {
            DWORD   PhysicalAddress;
            DWORD   VirtualSize;
    } Misc;
    DWORD   VirtualAddress;
    DWORD   SizeOfRawData;
    DWORD   PointerToRawData;
    DWORD   PointerToRelocations;
    DWORD   PointerToLinenumbers;
    WORD    NumberOfRelocations;
    WORD    NumberOfLinenumbers;
    DWORD   Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;

#define IMAGE_SIZEOF_SECTION_HEADER          40
```

Back with our example of pbrush.exe the .text section header has the following values

```
SECTION HEADER #4
  .reloc name
      34 virtual size
    4000 virtual address
    1000 size of raw data
    4000 file pointer to raw data
       0 file pointer to relocation table
       0 file pointer to line numbers
       0 number of relocations
       0 number of line numbers
42000040 flags
         Initialized Data
         Discardable
         (no align specified)
         Read Only
```

In order to append a new section to the file we need to look at the current last section, the .reloc section. The *virtual size* and the *size of raw data* of the section we are copying will remain the same but we will need to modify the *virtual address* and the *file pointer to raw data*. The *file pointer to raw data* is simply the current end of file (0x5000) which should be equal to the *size of raw data* plus the *file pointer to raw data* of the current last section. The *virtual address* is more complex since we need to round it up as follows. Add the *virtual size* to the *virtual address* of the reloc section (0x4034) and round up to the next 0x1000 boundary to get 0x5000. This is the *virtual address* of the added section.

In the NT header you will find a value giving the number of sections. This will now need to be increased by one. You also need to find the address of the current last section header. In most cases there is plenty of space to add several new section headers. The pbrush.exe file only has four sections so a new section header can easily be inserted. The new section is added 0x40 bytes (IMAGE_SIZEOF_SECTION_HEADER) after the address of the .reloc section. Add this by overwriting the existing bytes (they are all zero). Finally change the name of the original copy of the section to something else (it doesn't really matter what)

Thats it. There are now two identical copies of the original section. The CD protection system will check the copy appended to the end of the file and you are free to apply patches to your hearts content to the original and executing, section. (Note that the copy is never actually executed it is only ever read to calculate it's check-sum value so you don't need to patch it).

Final Notes

The PE format is quite complex especially when you start looking at the formats of the various section types, .reloc, .idata, .edata etc. It becomes even more difficult when propriatory information is appended to the end of the exe file. Delphi for example seems to hold most of its code appended to the end of the exe file, one of my new projects will be to look at how this information is organised. The techniques I described here will get you started in understanding the PE format. There is more that you (and I) can find out, but at least this is a start.

Oh, by the way, if you were wondering what the target was that I cracked using the section copy technique it is any title protected with SafeDisc.

Ob Duh

I wont even bother explaining you that you should BUY this target program if you intend to use it for a longer period than the allowed one. Should you want to STEAL this software instead, you don't need to crack its protection scheme at all: you'll find it on most Warez sites, complete and already regged, farewell, don't come back.