Push the Red Button

Malware, encryption, reverse engineering, networking, and other arcana.

Sunday, August 26, 2007

PDB Stream Decomposition

I've been taking a look at the structure of PDB files recently. PDB files are Microsoft's proprietary file format for storing debug information. They are generated by the Visual Studio family of products, and Microsoft has been kind enough to provide a full set of PDB files for its own operating system files since Windows 2000, which means that we get access to a whole bunch of great information like function symbols and type information. These can provide an excellent insight into the internals of the Windows operating system.

Naturally, the file format is undocumented and proprietary (there used to be a page on MSDN stating as much, but I can't seem to find it now). However, Sven Schreiber, in his book *Undocumented Windows 2000 Secrets: A Programmer's Cookbook*, provides details on the format and some sample code for reading the information in such files. Although the original code only deals with PDB files created with Visual Studio 6.0 and below, he has recently released a new program that parses the most recent version of the file format. As the newer format has not been written about, I thought I'd devote this entry to describing its high-level structure, and provide some code for breaking it down into its component parts. Hopefully I'll be following it up next week with some more polished code to parse the internal format of several of the streams (such as stream 2, which contains type information), but as I'm going to be moving this week, it may get pushed off until the week after.

(By the way, in case you're wondering why I don't simply use the Debug Interface Access SDK to get at this information, the main reason is that it's not available for my preferred operating systems, Linux and OS X. I also think it's good in general to know what's really going on inside a file format; it gives you a better feel for how reliable the information presented is (as opposed to just believing what the tool tells you), and can really save you when you find that the tool you use has some limitations or doesn't do what you expect.)

PDB files start with a fairly long, 0x20 byte signature:

```
'Microsoft C/C++ MSF 7.00\r\n\x1ADS\0\0\0'
```

Next come several dwords that give vital information and allow some basic integrity checking (note: I'm using Schrieber's names here):

- dPageBytes - the size of a page in bytes.
- dFlagPage - page number of the table describing which pages are allocated in the file. If page *n* of the PDB file is in use, the *n*th bit of the page will be set to 0.
- dFilePages - number of pages in this file. dFilePages * dPageBytes should equal the file's size.
- dRootBytes - the size of the root stream, in bytes.
- dReserved - always zero.
- adIndexPages[] - an array of dwords giving the pages numbers that contain the root stream index. In fact, I do not have any evidence that this is an array in the latest PDB format, rather than a simple dword, but Schreiber lists it as such.

All data in the PDB file is stored in blocks of a fixed size given in the dPageBytes member of the header. As a result, most offsets in the file are given as a page number, rather than a byte offset. To get the actual file offset, simply multiply the page number by dPageBytes. Overlaid on top of this block structure, information is stored in *streams*, much like files on a filesystem. A single structure, the *root stream*, provides an index of the streams, and on what pages they may be found. If this structure sounds familiar, that's because it appears to be quite a common one within Microsoft-this general layout is shared with the FAT filesystem, OLE (e.g. Word) documents, and Windows registry files.

If we seek to the page referred to by adIndexPages, we find that it contains an array of dwords giving the page numbers that make up the root stream. However, it is not immediately obvious how we are to know the length of this array. In fact, we must calculate it: since we know the size of the root stream from dRootBytes, and the size of a page from dPageBytes, the number of pages required to store the root stream is given by dRootBytes / dPageBytes (round up). For example, in the PDB file for ntoskrnl.exe under

Contact

Email: brendandg@nyu.edu (PGP)
Twitter: @moyix
Homepage: http://engineering.nyu.edu/people/brendan-dolan-gavitt

Promiscuous Linkage

- My GT Home Page
- Offensive Computing
- Penny Arcade
- SANS Internet Storm Center
- Computer Forensic Blog
- Volatile Systems Blog
- Volatility Tumbleblog
- Windows Incident Response
- XKCD

Windows XP SP2, dRootBytes is 15964, and the page size is 0x400 bytes, so the number of pages (and hence the length of the root stream index array) is 16. We can now read each page listed in the array to reconstruct the full root stream.

The root stream itself has a fairly simple structure. It starts with a dword, dStreams, giving the number of streams present in the file. Next is an array of dwords of length *dStreams*, giving the length of each stream in bytes. Finally, there are *dStreams* arrays of dwords listing the pages that make up each stream. In other words, the structure looks like:

```
[dStreams][length of stream 0][length of stream 1]...[length of stream n][first page of stream 0][second page of stream 0]...[last page of stream 0][first page of stream 1][second page of stream 1]
```

...and so on.

Once again, we have to calculate the size of each array that lists the page numbers for each stream. Luckily, it works exactly as before: the length of each array is given by the size of the stream divided by the page size, round up. Now that we have the list of pages that make up each stream, it is a simple matter to write out each stream separately.

Code to do just this can be found here (note: this may go down for a few days next week, as I move things to my new apartment. If anyone knows of a good shared host so I can stop hosting things off my desktop, let me know!) The usual caveats apply: use at your own risk, alpha quality code, and no support for older PDB files (some of the XPSP2 debug symbols use this format; it looks like all of the Vista symbols use the new format, however).

That's all for now! Next time, I'm hoping to present the beginnings of a full-fledged PDB parser, including support for the internal structures of some streams, such as the type information (stream 2) and function symbols (stream 3). Personally, the type information is most interesting to me; extracting this data from ntoskrnl.exe (ntkrnlmp.exe on Vista) allows us access to details of a number of internal kernel structures, which we can then use to interpret the contents of volatile memory. I used this technique with excellent results in my investigation of the VAD tree. Eventually, I hope to be able to output information on Windows kernel structures in a format that can be inserted into Volatility's vtypes.py.

Posted by Brendan Dolan-Gavitt at 6:21 PM
Labels: debugging, file formats, pdb, undocumented, windows

No comments:

Post a Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)