HP Security Research Blog

The HP Security Research blog provides a platform for security experts from across HP to discuss innovative research, industry observations, and updates on the threat landscape to help organizations proactively identify and manage risk.

Share

PDB Type Theft

Dustin_Childs | October 12, 2015

Post a Comment

In August of this year, Microsoft published an update to NTDLL and along with it, released updated symbols for debugging. These symbols are available as PDBs (program databases). Unfortunately, the symbols that were released contain type information that is missing standard structures and enumerations. As a result, debugging applications on Windows became a far more involved task. Microsoft is aware of the issue but has yet to release updated PDBs that rectify this issue. While they are working on it, I found myself wondering if I could avoid their involvement altogether. Barring any changes to the structures and enumerations, the information from previous versions of the PDBs should still be valid. As such, if I could copy the type information from a previous PDB and inject it into the current PDB, I'd theoretically be able to have everything I expect from a working build process.

So that is what I did.

I initially started with pdbparse, an open-source Python module for parsing Microsoft PDB files. Another tool I considered is pdbparser, which is written in C/C++ and seems a bit more complete. Both have their advantages but neither quite fit my needs. I ended up implementing my own version in a poor attempt to make the code a little cleaner for both reading and writing PDBs.

Fortunately, the PDB file format is simple enough that I was able to do this with relative ease. A PDB is broken up into pages of data where the size of a page is specified within the header. As one can imagine, the header is in the first page of the file, and it describes the location of the pages for the Root stream. The Root stream then specifies the number of streams, the sizes of the respective streams, and the pages the streams occupy. There are a couple of hard-coded indexes within the streams. For example, the first stream described within the Root stream is always a copy of the Root stream. The stream we are primarily interested in is the third stream, which describes type information. The header for the type information stream describes another stream named TpiHash. I do not yet understand the contents of the values; only that they are required for type information to be used.

Most of the code within the script is responsible for parsing the structures as part of reading a PDB and for preparing structures as part of writing a PDB. The code to handle copying the type information is pretty straightforward and ultimately boils down to the following:

- Open the PDB containing the type information to copy.
- Open the PDB that is to receive type information.
- Replace the type information stream from the PDB in #1 to the PDB in #2.
- Find the TPIHash stream from PDB #2 and overwrite it with the TPIHash stream from PDB #1.
- Update the TPI stream to point to the TPIHash stream.
- Write out the updated PDB.

You can find the script itself, dubbed pdb_type_theft.py, here. Usage is straight-forward -- just call the script with the first argument as the PDB to copy types from and the second argument as the PDB to dump types into.

This script requires having a PDB with the type information you want available to copy into another PDB. If you are not in the habit of snapshotting your VMs after every update, the following links may be helpful -- just make sure to set your user-agent to 'Microsoft-Symbol-Server/6.6.0007.5' when downloading them. Note that these images will need to be extracted with a tool that can handle .cab files. Windows 7 can extract them if you rename the file to end with a .cab extension.

Windows 7 32-bit:

My Community

Getting Started

Join the Conversation

Register Now

Already a member?

Log in

Quick Links

Search

Blog

Search HP Forum Go

About the Author

Dustin_Childs

I am a senior security content developer with Hewlett-Packard Security Research (HPSR). In this role, I write and edit security analysis and...

Top Kudoed Posts

The monthly patch review – September, 2015	1

View All

Featured

Follow Us

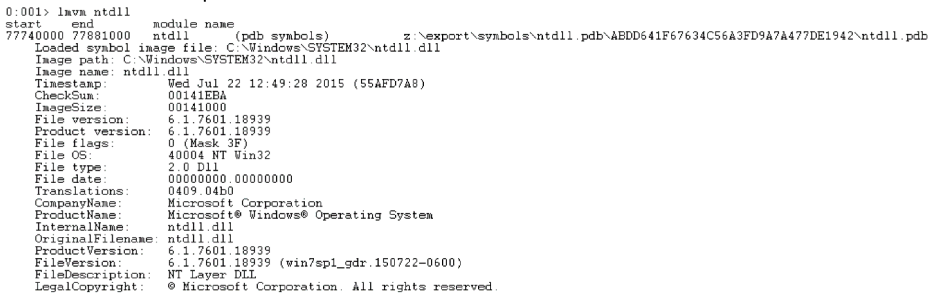
Latest Articles

- The monthly patch review – October, 2015
- PDB Type Theft
- HP Security Research OSINT (OpenSource Intelligenc...
- Security intelligence viewpoint: Encryption as an ...
- HP Security Research OSINT (OpenSource Intelligenc...
- HPSR Software Security Content 2015 Update 3
- HP Security Research OSINT (OpenSource Intelligenc...
- HP Security Research OSINT

http://msdl.microsoft.com/download/symbols/ntdll.pdb/6610BBDECCA44BA5B080A03FA694E08C2/ntdll.pd_
http://msdl.microsoft.com/download/symbols/ntkrpamp.pdb/B58467BE7C8749778B01CA322F7CE7152/ntkrpamp.p...

Windows 7 64-bit:
http://msdl.microsoft.com/download/symbols/ntdll.pdb/062792552C5C4636905651E175CDD7182/ntdll.pd_
http://msdl.microsoft.com/download/symbols/wntdll.pdb/B081677DFC724CC4AC53992627BEEA242/wntdll.pd_
http://msdl.microsoft.com/download/symbols/ntkrnlmp.pdb/C1CA6E0E486D490DBAA23F09BED5712B2/ntkrnlmp.p...

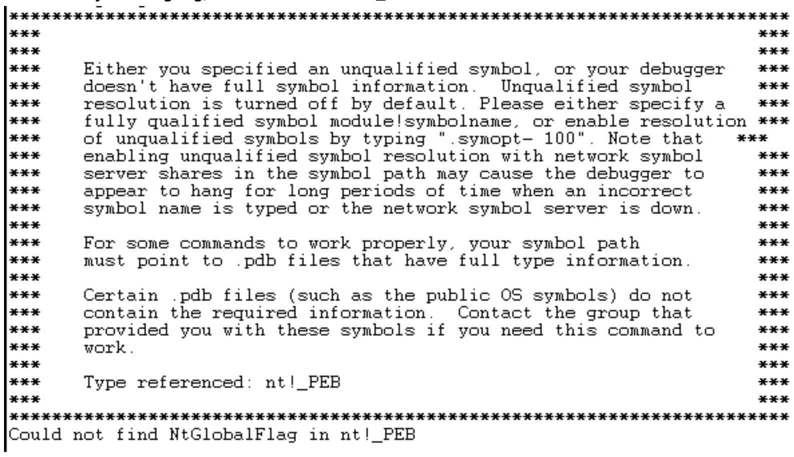
Here is the current (incomplete) module information:



```
0:001> lmvm ntdll
start    end        module name
77740000 77881000   ntdll      (pdb symbols)          z:\export\symbols\ntdll.pdb\ABDD641F67634C56A3FD9A7A477DE1942\ntdll.pdb
    Loaded symbol image file: C:\Windows\SYSTEM32\ntdll.dll
    Image path: C:\Windows\SYSTEM32\ntdll.dll
    Image name: ntdll.dll
    Timestamp:        Wed Jul 22 12:49:28 2015 (55AFD7A8)
    CheckSum:         00141EBA
    ImageSize:        00141000
    File version:     6.1.7601.18939
    Product version:  6.1.7601.18939
    File flags:       0 (Mask 3F)
    File OS:          40004 NT Win32
    File type:        2.0 Dll
    File date:        00000000.00000000
    Translations:     0409.04b0
    CompanyName:      Microsoft Corporation
    ProductName:      Microsoft® Windows® Operating System
    InternalName:     ntdll.dll
    OriginalFilename: ntdll.dll
    ProductVersion:   6.1.7601.18939
    FileVersion:      6.1.7601.18939 (win7sp1_gdr.150722-0600)
    FileDescription:  NT Layer DLL
    LegalCopyright:   © Microsoft Corporation. All rights reserved.
```

Figure 1 - NTDLL Module Information

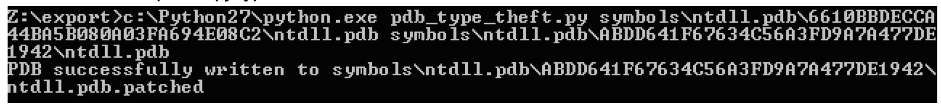
Now we'll try to run !gflag, which references the _PEB structure:



```
*************************************************************************
***                                                                   ***
***                                                                   ***
***    Either you specified an unqualified symbol, or your debugger   ***
***    doesn't have full symbol information.  Unqualified symbol      ***
***    resolution is turned off by default. Please either specify a   ***
***    fully qualified symbol module!symbolname, or enable resolution ***
***    of unqualified symbols by typing ".symopt- 100". Note that   ***
***    enabling unqualified symbol resolution with network symbol     ***
***    server shares in the symbol path may cause the debugger to     ***
***    appear to hang for long periods of time when an incorrect      ***
***    symbol name is typed or the network symbol server is down.     ***
***                                                                   ***
***    For some commands to work properly, your symbol path           ***
***    must point to .pdb files that have full type information.      ***
***                                                                   ***
***    Certain .pdb files (such as the public OS symbols) do not      ***
***    contain the required information.  Contact the group that      ***
***    provided you with these symbols if you need this command to    ***
***    work.                                                          ***
***                                                                   ***
***    Type referenced: nt!_PEB                                       ***
***                                                                   ***
*************************************************************************
Could not find NtGlobalFlag in nt!_PEB
```

Figure 2 - Output from !gflag

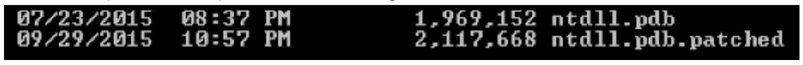
Next we run the script to copy type information stream from the PDB:



```
Z:\export>c:\Python27\python.exe pdb_type_theft.py symbols\ntdll.pdb\6610BBDECCA
44BA5B080A03FA694E08C2\ntdll.pdb symbols\ntdll.pdb\ABDD641F67634C56A3FD9A7A477DE
1942\ntdll.pdb
PDB successfully written to symbols\ntdll.pdb\ABDD641F67634C56A3FD9A7A477DE1942\
ntdll.pdb.patched
```

Figure 3 - Script execution

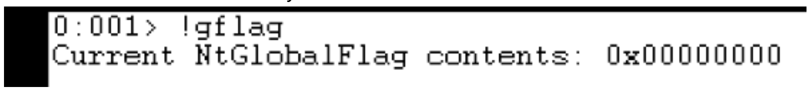
Here is a comparison of the patched file versus original file:



```
07/23/2015  08:37 PM         1,969,152 ntdll.pdb
09/29/2015  10:57 PM         2,117,668 ntdll.pdb.patched
```

Figure 4 - File comparison

Then after we rename it and reload symbols:



```
0:001> !gflag
Current NtGlobalFlag contents: 0x00000000
```

Figure 5 - Successful load of symbols

Success!

That script was also tested against the symbols for ntoskrnl.exe since they have the exact same issue. One caveat with the script is that it does not support the older PDB2 format. It may also have trouble with different versions of the TPI stream. Modifying the script to support those versions would likely be doable, however I focused solely on fixing the issue with the Microsoft symbols. Hopefully this helps makes debugging on Windows 7 a little easier until Microsoft manages to officially fix the symbols.

Jasiel "WanderingGlitch" Spelman

Tags: debugging| HPSR| ZDI
Labels: debugging| HPSR| ZDI
Everyone's Tags: View All (3)
debugging HPSR ZDI

(OpenSource Intelligenc...
Security research survey: What's your OpSec situat...
HP Security Research OSINT (OpenSource Intelligenc...

Latest Comments

SBradley on: HP Security Briefing, Episode 22: The hidden dange...
SasiSiddharth on: Analyzing CVE-2015-1635 from cause to cure
Percy Rotteveel on: Crypto Manifesto 2015
Ivan Sh on: Full details on CVE-2015-0096 and the failed MS10-...
Sandeep Sagar on: HPSR, Microsoft, disclosure, and the $125,000 bug ...
Giorgio di Grazia on: POS malware - a look at Dexter and Decebal
Nicholas_Fagan on: Technical analysis of the SandWorm Vulnerability (...
Ajayi Oluwaseun Emmanuel on: Hacking my smart TV - an old new thing
alvaro_munoz on: Update your Struts 1 ClassLoader manipulation filt...
Lotie Brink on: Reverse Engineering NAND Flash Memory – POS device...

Archives

October 2015
September 2015
August 2015
July 2015
View Complete Archives

Labels: debugging HPSR ZDI

Share

0

Post a Comment Permalink

Leave a Comment

We encourage you to share your comments on this post. Comments are moderated and will be reviewed and posted as promptly as possible during regular business hours

To ensure your comment is published, be sure to follow the Community Guidelines.

*Name

*Email

Website (optional)

http://

Rich Text | Preview

Quote

Font Family Font Sizes

*Word verification by reCAPTCHA

Get a new challenge Get a sound challenge Help with word verification

Cancel Post Your Comment