The VAD Tree: A Process-Eye View of Physical Memory

By

Brendan Dolan-Gavitt

From the proceedings of

The Digital Forensic Research Conference

DFRWS 2007 USA

Pittsburgh, PA (Aug 13th - 15th)

available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/diin

Digital Investigation

The VAD tree: A process-eye view of physical memory☆

Brendan Dolan-Gavitt

MITRE Corporation, 202 Burlington Road, Bedford, MA, United States

Keywords:
Virtual Address Descriptors
Digital forensics
Microsoft Windows
Volatile memory
Anti-forensics

ABSTRACT

This paper describes the use of the Virtual Address Descriptor (VAD) tree structure in Windows memory dumps to help guide forensic analysis of Windows memory. We describe how to locate and parse the structure, and show its value in breaking up physical memory into more manageable and semantically meaningful units than can be obtained by simply walking the page directory for the process. Several tools to display information about the VAD tree and dump the memory regions it describes will also be presented.

1. Introduction

The task of analyzing a dump of physical memory for signs of an intrusion or malicious code can be a daunting process. It is not uncommon to have to deal with systems with several gigabytes of physical memory, and tools and research in volatile memory forensics are still relatively in early stages compared to disk forensics. When dealing with such volumes of data, it is often helpful to employ an additional layer of abstraction to summarize and give context to the information at lower levels. We propose that the Virtual Address Descriptor (VAD) tree, a kernel data structure found in Windows memory, can provide such an abstraction layer over the page directory and page tables by describing the memory ranges allocated by a process as they might be seen by the process – as mapped files, loaded DLLs, or privately allocated regions. We will also show how the VAD tree can be used to list DLLs loaded by a process, detect certain types of library injection, and defeat DLL hiding methods such as those used by NTIllusion (Kdm, 2004).

2. Related work

A great deal of attention has recently been focused on physical memory analysis. In 2005, the winners of the DFRWS Memory Analysis Challenge focused on finding processes and threads in Windows memory using techniques such as walking the linked list process table (Garner and Mora; Betz). Schuster (2006a) later developed a signature-based method of identifying processes and threads, which is capable of finding processes that had been intentionally removed from such linked lists, as well as those that had exited – sometimes even after the system had rebooted. Since then, further work (Walters and Petroni, 2007; Schuster, 2006b) has demonstrated techniques for finding useful objects such as open sockets and TCP connections in memory.

The VAD structure itself is not entirely undocumented. It is briefly mentioned by Dabak et al. (1999); however, the authors only document one of the three types of node in the tree, and some important details, such as the meaning of the Flags field, are left out. Russinovich and Solomon (2004) also describe the VAD tree, but only give details on how it is used in the memory manager, not its actual structure in memory. Most of the information in this paper was derived using the debug symbols for Windows XP Service Pack 2. Older versions of Windows do not appear to contain the symbols for this particular data structure.

3. The VAD tree

The Virtual Address Descriptor tree is used by the Windows memory manager to describe memory ranges used by

☆ The author's affiliation with The MITRE Corporation is provided for identification purposes only, and is not intended to convey or imply MITRE's concurrence with, or support for, the positions, opinions or viewpoints expressed by the author.

E-mail address: brendandg@mitre.org

1742-2876/$ – see front matter © 2007 DFRWS. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2007.06.008

a process as they are allocated. When a process allocates memory with VirutalAlloc, the memory manager creates an entry in the VAD tree. The corresponding page directory and page table entries are not created until the process tries to reference that memory page, which can provide significant memory savings for processes that allocate a large amount of memory but access it sparsely (Russinovich and Solomon, 2004, pp. 448–449). The tree itself is a self-balancing binary tree: at any given node, memory addresses lower than those contained at the current node can be found in the left subtree and higher ranges in the right (Fig. 1).

There are three distinct types of VAD node found in Windows 2000 and XP. The structures are named _MMVAD_SHORT, _MMVAD, and MMVAD_LONG, and their format can be determined using WinDbg and Microsoft's debugging symbols under Windows XP SP2. The structures do not appear to have changed significantly since Windows 2000 SP4. Each structure builds off the one before it, so an _MMVAD is simply an _MMVAD_SHORT with several extra fields, such as ControlArea and VadFlags2. The only reliable mechanism currently known to distinguish the three structures during parsing is to examine the pool tag[1]: _MMVAD_SHORT structures are tagged with "VadS", _MMVADs with "Vad", and _MMVAD_LONGs with "Vadl".[2]

Walking the tree is simply a matter of identifying the _EPROCESS structure for the process of interest (using, for example, Schuster's (2006a) *PTfinder*), locating the VadRoot member (offset 0×194 in Windows 2000 SP4, 0×11c in all versions of XP), and then following each link to the left and right subtrees until the entire tree is traversed. All addresses are virtual, so the page directory for the process is also needed in order to successfully read the tree.[3]

VAD nodes also reference a number of other kernel structures that may be useful to an investigator. For _MMVADs and _MMVAD_LONGs, one can find the address of the _CONTROL_AREA for the memory region.[4] From there, if the region is used for a mapped file (such as a loaded DLL), the corresponding _FILE_OBJECT structure can be referenced and the name of the file can be extracted. This can provide an investigator with an alternate method of listing modules loaded by a process, which can then be compared with other methods (e.g., those described by Walters, 2006) to find evidence of library injection and hidden modules.

4. Tools and results

We have developed several tools to allow forensic investigators to examine the VAD tree and dump the memory regions described out to disk. vadwalk.py walks the entire VAD tree and checks its integrity, and can display the tree in several formats: a tabular listing of each node, an ASCII-art representation, or a Graphviz dotfile. vadinfo.py Prints detailed information about each node in the VAD tree of a process, including any associated ControlAreas or FileObjects. Finally, vaddump.py walks the tree and writes each memory range to disk. The tools work with VMWare vmem images and raw memory dumps acquired using dd, and support dumps from Windows 2000 SP4 and all versions of Windows XP.

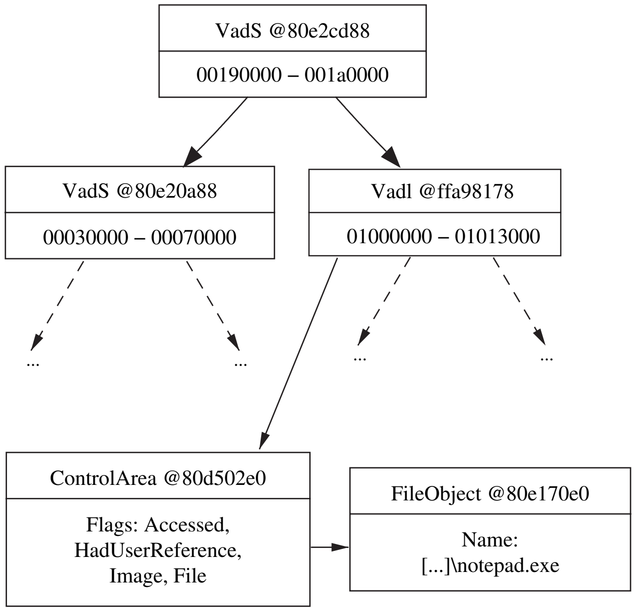

Fig. 1 – A portion of the VAD tree for notepad.exe.

Although the tools are still at an early stage, they can already be used to easily detect some types of malicious library injection. For example, the "user-mode rootkit" NTIllusion (Kdm, 2004) is capable of injecting libraries into processes and then hiding the injected DLLs by unlinking them from the module list stored in the Process Environment Block (PEB). Such modification of the PEB, however, does not affect the VAD tree, and the loaded DLL can be seen as a mapped file in the address space of the affected process using vadinfo; such evidence cannot be found with tools that simply walk the PEB's module list.

The VAD tree can also provide useful information to an investigator who already has some idea of what to look for: if there is a particular string that is known to be an indicator of malicious activity (for example, a domain name that a trojan uses to "phone home"), the memory space of each process could be dumped, and a string search then performed on the output using a tool like grep or FTimes. Such attribution of strings to user-space processes is already possible by examining the page directory for the process, and tools such as AAron Walters' Volatools (Walters and Petroni, 2007) and Joe Stewart's *pmoddump* implement this functionality. By looking at the VAD tree, however, an examiner can also say whether such strings occurred in a loaded DLL, a dynamically allocated memory region, or the process executable itself.

[1] For more information on pool tags and pool allocations in general, consult Schuster (2006b).

[2] There is, in fact, one slight complication – Windows 2000 does not use the "Vadl" tag, but rather indicates that the structure is an _MMVAD_LONG by setting the "LongVad" flag in the VadFlags2 field and using the "Vad " tag.

[3] The process of translating virtual addresses will not be described here; the procedure is well-documented in Kornblum (2007), Intel®Corporation (2006).

[4] Additional information on Control Areas can be found in Russinovich and Solomon (2004, pp. 453–457).

5. Anti-forensic attacks

As with other kernel data structures, the VAD tree can be altered by any code running in ring 0. This means that Direct Kernel Object Manipulation (DKOM) attacks can be carried out to unlink VAD nodes from the tree and effectively hide them from forensic techniques that rely on this data structure. For example, a process can allocate a region of memory using VirtualAlloc and then reference it, forcing the memory manager to create the corresponding page table entries. The VAD node that describes the region can then be unlinked; the memory will no longer be visible from tools such as `vadwalk`, but the process will still be able to access it normally (memory reads appear to use the page directory to access memory first, and the VAD is only consulted if a page fault occurs (Russinovich and Solomon, 2004, p. 448)).

We have successfully tested this attack on Windows XP by allocating several pages of memory, filling them with a specified value, removing the appropriate VAD entry with WinDbg, and finally reading the memory from within the process to verify that it was still accessible. Analysis of the process's VAD tree in a physical memory dump did not find the memory region that had been hidden. Further research is needed to determine what other attacks against the VAD may be possible and explore ways of mitigating them.

6. Future work

At the moment, the VAD tree is only available in processes that are still running; the Windows kernel appears to zero out the pointer to the VAD root when the process exits. This limits the usefulness of the technique for examining many of the processes found by tools such as *PTfinder*, which can locate processes that are no longer directly referenced by the kernel. One avenue of future investigation would be to attempt to find a way to recover the unlinked trees in these cases, allowing access to more detailed information about processes that are no longer running.

The tools themselves also need significant work to move them beyond the proof-of-concept stage. In Windows Server 2003 and Windows Vista, the structure of the VAD tree appears to have changed (Schuster, 2007), and the tools will need to be updated to reflect the new structure.[5] The VAD tools should also be integrated with an existing volatile memory forensics framework, such as FatKit (Petroni et al., 2006), which will allow investigators to leverage features like support for invalid page table entries, as documented by Kornblum (2007), and thereby get a more complete view of physical memory.

7. Conclusion

We have shown that examination of the VAD tree can provide useful information to an investigator such as the pattern of memory allocations used by a process and files mapped into its virtual address space. It is also another entry point into the complex web of linked data structures that allow the kernel to manage processes, threads, and other objects in Windows. Although it presently has some limitations, such as the inability to give information about processes that had exited at the time of the memory dump, we feel that when used in conjunction with other memory analysis techniques, the VAD tree can shed new light on the behavior of user-space processes.

Acknowledgements

Many thanks go out to Aaron Walters, who read over drafts of this paper and gave many great suggestions, as well as Andy Bair, who got me interested in the topic of digital forensics in the first place.

References

Betz C. DFRWS 2005 forensic challenge answers, <http://www.dfrws.org/2005/challenge/ChrisBetz-DFRWSChallengeOverview.html>.

Dabak P, Phadke S, Borate M. Undocumented Windows NT. New York, NY, USA: John Wiley & Sons, Inc., ISBN 0764545698; 1999.

Graphviz, <http://www.graphviz.org/>.

Garner Jr GM, Mora R-J. DFRWS 2005 forensic challenge answers, <http://www.dfrws.org/2005/challenge/RossettoeCioccolato-Responses.pdf>.

Intel®Corporation. Intel®64 and IA-32 architectures software developer's manual. Santa Clara, CA, USA: Intel®Corporation, <http://www.intel.com/products/processor/manuals/index.htm>; 2006.

Kdm. NTIllusion: a portable Win32 userland rootkit. Phrack July 2004;11(62).

Kornblum J. Using every part of the buffalo in Windows memory analysis. Digit Investig J, <http://jessekornblum.com/research/papers/buffalo.pdf> March 2007.

Petroni Jr NL, Walters A, Fraser T, Arbaugh WA. FATKit: a framework for the extraction and analysis of digital forensic data from volatile system memory. Digit Investig December 2006;3(4).

Russinovich ME, Solomon DA. Microsoft Windows internals, Microsoft Windows Server(TM) 2003, Windows XP, and Windows 2000 (Pro-Developer). 4th ed. Redmond, WA, USA: Microsoft Press, ISBN 0735619174; 2004.

Schuster A. Searching for processes and threads in Microsoft Windows memory dumps. In: Proceedings of the sixth annual digital forensic research workshop (DFRWS 2006); 2006a. <http://www.dfrws.org/2006/proceedings/2-Schuster.pdf>.

Schuster A. Pool allocations as an information source in Windows memory forensics. In: Third international conference on IT-incident management & IT-forensics; October 2006b.

Schuster A. _EPROCESS version 6.0.6000.16386, <http://computer.forensikblog.de/en/2007/01/eprocess_6_0_6000_16386.html>; January 2007.

The FTimes project, <http://ftimes.sourceforge.net/FTimes/index.shtml>.

Walters A. FATKit: Detecting malicious library injection and upping the "anti". Technical report. 4TΦ Research Laboratories; July 2006.

Walters A, Petroni Jr NL. Volatools: integrating volatile memory forensics into the digital investigation process. In: Black Hat DC 2007; 2007.

Brendan Dolan-Gavitt received a BA in Computer Science and Mathematics from Wesleyan University in Middletown, CT in 2006. He currently works at the MITRE Corporation in Bedford, MA.

[5] This change may correspond to the move to AVL self-balancing binary trees, as documented in Russinovich and Solomon (2004).