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

By
Brendan Dolan-Gavitt

Presented At
The Digital Forensic Research Conference
DFRWS 2007 USA Pittsburgh, PA (Aug 13th - 15th)

DFRWS is dedicated to the sharing of knowledge and ideas about digital forensics research. Ever since it organized the first open workshop devoted to digital forensics in 2001, DFRWS continues to bring academics and practitioners together in an informal environment. As a non-profit, volunteer organization, DFRWS sponsors technical working groups, annual conferences and challenges to help drive the direction of research and development.

http://dfrws.org
The VAD Tree: A Process-Eye View of Physical Memory

Brendan Dolan-Gavitt
Too Much Data

- Physical memory is voluminous, and getting more so
- Structure not well-understood for most operating systems (e.g. Windows)
- Want to give context to artifacts and evidence found in memory dumps
Current Approaches

- Attribution: look at the PDEs and PTEs for each process to determine what owned a particular bit of memory (e.g., Volatility’s “strings” module)

- Find interesting structures such as processes, threads, kernel modules, TCP connections, etc. in memory
The VAD Tree

- Virtual Address Descriptor tree is a self-balancing binary tree that lists memory ranges allocated using VirtualAlloc().

- Three variations on the structure: short, normal, and long (_MMVAD_SHORT, _MMVAD, _MMVAD_LONG in XP SP2 debug symbols).
Walking the VAD Tree

1. Find an _EPROCESS structure (using, e.g., Andreas Schuster’s PTFinder).

2. Read the VadRoot member of that structure.

3. Use the pool tag (see Schuster IMF 2006) to determine the type of VAD node, then visit the left and right children.

4. GOTO 3.
Useful Properties

• VAD nodes correspond to the amount of memory requested by VirtualAlloc, rather than 4096 byte chunks as in page directory.

• Normal and long VADs contain pointers to Control Areas, which in turn point to File Objects if the memory range corresponds to a mapped file such as a DLL.

• Kernel memory structure—harder to modify.
VAD Tools

- **vadwalk**: walk the tree and give a short listing as ASCII art, table view, or GraphViz
- **vadinfo**: print detailed information on each node
- **vaddump**: write the memory ranges described by each node out to disk
Tool Limitations

- Naïve memory model (no support for “invalid” PDE/PTEs, such as prototype or transition pages).
- 32-bit, non-PAE mode is assumed.
- Only Windows 2000 and XP (SP0, SP1, SP2) supported—no Vista support.
- Code quality is “proof-of-concept”—expect crashes on strange inputs.
Anti-Forensics

- VAD Tree susceptible to DKOM attacks.
- Once the memory has been committed and the page directory entries created, VAD does not appear to be used.
- Code with access to kernel memory could remove a node from the tree without affecting the user-space process’s ability to access it.
Future Work

- Vista VADs
- Finding and reconstructing VAD trees from exited processes (VadRoot member is zeroed out when process exits).
- Other interesting structures pointed to—for example, what are _MMBANKED_SECTION and _MMEXTEND_INFO?
Links

• **Volatility** – GPL-licensed memory analysis framework by AArnon Walters and Nick Petroni, Jr.

• **VAD Tools** – Public domain proof-of-concept tools.
Questions?