available at www.sciencedirect.com

ScienceDirect

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

Digital Investigation

Windows operating systems agnostic memory analysis

*James Okolica**, *Gilbert L. Peterson*

Department of Electrical and Computer Engineering, Air Force Institute of Technology, USA

Keywords:
Memory forensics
Microsoft windows
PDB files
Operating system discovery
Processes
Registry files
Network activity

ABSTRACT

Memory analysis is an integral part of any computer forensic investigation, providing access to volatile data not found on a drive image. While memory analysis has recently made significant progress, it is still hampered by hard-coded tools that cannot generalize beyond the specific operating system and version they were developed for. This paper proposes using the debug structures embedded in memory dumps and Microsoft's program database (PDB) files to create a flexible tool that takes an arbitrary memory dump from any of the family of Windows NT operating systems and extract process, configuration, and network activity information. The debug structures and PDB files are incorporated into a memory analysis tool and tested against dumps from 32-bit Windows XP with physical address extensions (PAE) enabled and disabled, 32-bit Windows Vista with PAE enabled, and 64-bit Windows 7 systems. The results show the analysis tool is able to identify and parse an arbitrary memory dump and extract process, registry, and network communication information.

1. Introduction

Memory analysis is an integral part of effective computer forensics. Since the DFRWS memory challenge in 2005 (Digital Forensics Research Workshop, 2005), there has been significant research done in improving analysis of memory dump files (Betz, 2005; Schuster, 2006b; Walters and Petroni, 2007). Unfortunately, these techniques still rely on knowing characteristics of the operating system a priori. Furthermore, in most cases, these tools only work on a small number of operating system versions. For instance, while Volatility has extensive functionality, it only works on Microsoft Windows XP SP2 and SP3. What is needed is a tool that works on an arbitrary memory dump regardless of the operating system version and patch level.

This paper is a first step in achieving this generalized functionality. By incorporating the work of Alex Ionescu and Microsoft's program database (PDB) files (Microsoft Support) into a memory analysis tool, the tool is able to identify the operating system and version of a memory dump from the family of Microsoft NT operating systems (i.e., Windows NT4, Windows 2000, Windows Server 2003, Windows XP, Windows Vista, Windows Server 2008, and Windows 7). The tool then uses this information to locate the kernel executable and extract its globally unique identifier (GUID). With the kernel name and GUID, the tool retrieves the PDB file from Microsoft's online symbol server and uses it to enumerate the key operating system structures necessary to parse the memory dump.

The remainder of this paper presents an overview of the memory analysis work already done and a methodology for combining these different pieces of memory analysis and parsing to make a Windows agnostic tool. Finally, the paper discusses applying the resulting tool to a memory dump from a 32-bit Windows XP SP3 with physical address extensions enabled and disabled, 32-bit Windows Vista with physical

* *Corresponding author.*
E-mail addresses: jokolica@afit.edu (J. Okolica), gpeterson@afit.edu (G.L. Peterson).
1742-2876/$ – see front matter © 2010 Digital Forensic Research Workshop. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2010.05.007

address extensions enabled, and 64-bit Windows 7. In each case, the tool identifies the operating system version and memory layout, extract all of the process and registry information (including pages stored in page files and memory backed files), and extract network communication information.

2. Background

Live forensics examines the most volatile, and generally the most recent cyber artifacts. Process activity, configuration changes, and network communication occur constantly and by examining volatile memory, the most recent instances of each of these are captured. Furthermore, the kernel executables residing on disk may not mirror the code actually running in memory (particularly if malware programs have hooked them). Examining the operating system programs that are in memory provides the most accurate picture of what the operating system is actually doing.

Live response information investigators typically seek include:

- system data and time,
- logged on users and their authorization credentials,
- network information, connections, and status
- process information, memory and process-to-port mappings
- clipboard contents
- command history
- services, driver information
- open files and registry keys as well as hard disk images (Prosise et al., 2003).

While ideally, the method for collecting memory should not affect the operating system, if no collection method has been implemented a priori, options are limited. In these cases, the best method may be to use software tools that will impact the operating system as a part of collecting the image. There are two distinct approaches: starting a new collection process (Carvey, 2007) or inserting a collection driver into an existing kernel process. The traditional software collection method is to start a new process, such as Madiant's Memoryze, that does not use operating system application programmer interfaces (APIs) or graphical user interfaces (GUI) so that it has less system impact and is less likely to be subverted by an infected operating system. However, creating a new process still creates new process records, object tables, and device tables as well as allocates space within a portion of main memory. The alternative is adding a driver to an existing kernel process. The downside of this method is that it modifies the space for one of the processes that will be captured. This may later call into question whether other, unintended changes were made to that process' space as well, possibly tainting the results.

There are several tools that parse memory dumps and extract process information. Two of the early tools that scanned memory dumps to find processes were Chris Betz's memparser (Betz, 2005) and Andreas Schuster's ptfinder (Schuster, 2006a). In addition, Brendan Dolan-Gavitt has developed tools for extracting Windows registry information (Dolan-Gavitt, 2008). More recently, Aaron Walters and others have developed Volatility (Walters and Petroni, 2007) which in addition to finding processes and registry information, also finds the network and configuration information. Furthermore, Volatility 1.3 parses hibernation files. However, what all of these tools have in common is that they are limited to specific versions of specific operating systems, e.g., 32-bit versions of Windows XP SP2 and SP3. The reason for this is that since the data structures used by an operating system change from version to version, new versions of the software are needed each time. However, Barbarosa and Ionescu have provided a means of discovering from within a memory dump, the operating system version that was running (Barbarosa; Ionescu). We combine this with Schreiber's method for analyzing the program database files (Microsoft Support) generated when Microsoft compiles its code (Schreiber, 2001a,b) to create a Windows agnostic memory analysis tool.

3. Methodology

By combining work done by (Barbarosa; Dolan-Gavitt, 2008; Ionescu; Russinovich and Solomon, 2005; Schreiber, 2001a; Schuster, 2006a; Walters and Petroni, 2007), it is possible to take an arbitrary memory dump from one of the Windows NT family of operating systems (i.e., Windows NT4, Windows 2000, Windows Server 2003, Windows XP, Windows Vista, Windows Server 2008, and Windows 7) and parse it. This Windows agnostic approach provides several benefits. First, memory analysis tools no longer need to be coded to a specific operating system version and patch level; second, memory dumps that are acquired without operating system interaction (e.g., via direct memory access) may be parsed without interacting with either the operating system or a system administrator. Finally, as new versions and patch levels of operating systems are released, the existing memory analysis tools should continue to work. Fig. 1 shows the Windows agnostic memory analysis process.

First, using the work of Barbarosa and Ionescu, _DBGKD_DEBUG_DATA_HEADER64, _KDDEBUGGER _DATA64 and _DBGKD_GET_VERSION64 records are found and parsed to determine whether the dump comes from a 32-bit, 32-bit with physical address extensions enabled, or a 64-bit operating system. Using this information (Russinovich and Solomon, 2005), the kernel page directory table base is found. With this information and (Russinovich and Solomon, 2005), virtual addresses are parsed into physical addresses. Next, the base address of the kernel executable and of tcpip.sys are found from _DBGKD_DEBUG_DATA_HEADER64 directly and via PS_LOAD ED_MODULE_LIST respectively. By examining the debug section of these two portable executables (Microsoft Windows Hardware Developer Central), the globally unique identifier (GUID) and age are extracted and used to download the correct program database from Microsoft's symbol server (Microsoft Support). The PDB file is then parsed (Schreiber, 2001a), and the exported kernel data structures are extracted. With these data structures, it is possible to parse the memory dump without any hard-coded offsets (although the names of the structures (e.g., _EPROCESS) do still need to be