Digital Investigation 20 (2017) 23–33

Contents lists available at ScienceDirect

Digital Investigation

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

Research Summary

Memory forensics: The path forward

Andrew Case [a], Golden G. Richard III [b, *]

[a] New Orleans, LA, United States
[b] Center for Computation and Technology and Division of Computer Science and Engineering, Louisiana State University, Baton Rouge, LA, United States

ARTICLE INFO

Article history:
Available online 11 January 2017

Keywords:
Memory forensics
Computer forensics
Memory analysis
Incident response
Malware

ABSTRACT

Traditionally, digital forensics focused on artifacts located on the storage devices of computer systems, mobile phones, digital cameras, and other electronic devices. In the past decade, however, researchers have created a number of powerful memory forensics tools that expand the scope of digital forensics to include the examination of volatile memory as well. While memory forensic techniques have evolved from simple string searches to deep, structured analysis of application and kernel data structures for a number of platforms and operating systems, much research remains to be done. This paper surveys the state-of-the-art in memory forensics, provide critical analysis of current-generation techniques, describe important changes in operating systems design that impact memory forensics, and sketches important areas for further research.

Introduction

Traditional storage forensics comprises a set of techniques to recover, preserve, and examines digital evidence and has applications in a number of important areas, including investigation of child exploitation, identity theft, counter-terrorism, intellectual property disputes, and more. Storage forensics tools are focused primarily on "dead" analysis, typically using bit-perfect copies of storage media. From these copies, deleted files or file fragments are recovered, patterns of file access are determined, past web browsing activity is observed, etc. Over the past decade, a number of factors have contributed to an increasing interest in *memory forensics* techniques, which allow analysis of a system's volatile memory for forensic artifacts. These factors include a huge increase in the size of forensic targets, larger case back-logs as more criminal activity involves the use of computer systems, the use of forensics techniques in incident response to combat malware, and trends in malware development, where malware now routinely leaves no traces on non-volatile storage devices. Importantly, memory forensics techniques can reveal a substantial amount of volatile evidence that would be completely lost if traditional "pull the plug" forensic procedures were followed. This evidence includes lists of running processes, network connections, fragments of volatile data such as chat messages, and keying material for drive encryption.

While there has been tremendous progress in building advanced memory forensics tools since the first rudimentary techniques were developed around 2004, much work remains to be done in this exciting research area. This paper surveys the state-of-the-art in a number of areas in memory forensics, including acquisition and analysis, and attempts to clearly lay out the research challenges that lie ahead. These challenges include not only work that remains to be done, such as better analysis techniques for user-level malware, but also fundamental shifts in how memory forensics tools are designed and how they operate, to accommodate significant changes in operating systems design.

Area of focus – memory acquisition

Historical approaches to memory acquisition

The ability to acquire volatile memory in a stable manner is the first prerequisite of memory analysis. Traditionally, memory acquisition was a straightforward process as operating systems provided built-in facilities for this purpose. These facilities, such as */dev/mem* on Linux and Mac OS X and \\.*Device**PhysicalMemory* on Windows, provided administrator-level users direct access to physical memory.

On modern systems, such facilities are generally not available, however, due to security concerns. In particular, both */dev/mem* and *PhysicalMemory* allowed for read and write access to physical memory (CrazyLord, 2002). This allowed malware not only to steal the contents of kernel memory, but also to modify it. The notorious

* Corresponding author.
E-mail addresses: andrew@dfir.org (A. Case), golden@cct.lsu.edu (G.G. Richard).

Phalanx2 rootkit (Case, 2012) leveraged */dev/mem* in this manner as do many other malware variants across operating system versions.

Beyond the security issues, these built-in facilities were becoming of limited use to investigators as systems under investigations were rapidly moving towards:

- Having multiple CPU cores and/or physical processors installed
- Increasing amounts of RAM
- Operating system adoption to the demands of scale

Even if the kernel devices such as */dev/mem* were available, the implementations are not safe for memory acquisition on multicore/multi-CPU systems, as races to map, remap, and unmap pages can result in kernel instability. Furthermore, since the acquisition tools reading from these interfaces are executing in userland, in-kernel synchronization primitives can't be used to solve the problem. Thus kernel modules must typically be loaded to allow memory to be acquired and on multicore/multi-CPU systems, the kernel modules must be carefully designed to pay special attention to the operating system-specific rules governing how and when kernel mode code can be interrupted (generally known as kernel preemption) (Stüttgen and Cohen, 2014).

The continuously increasing amount of RAM installed in systems leads to page smearing (Carvey, 2005), which is an inconsistency between what the state of memory as described by the page tables is versus what is actually in those pages of memory. This issue occurs due to the time lapse between when the page tables are acquired versus when data is acquired in other portions of RAM. Smearing will be discussed in full detail in the following sections.

Current issues – page smearing

The following sections describe current approaches to acquisition across all major operating systems, along with the limitations of these approaches. Each section is structured to describe the state of the art, its limitations, and future directions to improve acquisition techniques and procedures. We start with page smearing as it is one of the most pressing issues.

Current state

As mentioned above, page smearing is an inconsistency that occurs in memory captures when the acquired page tables reference physical pages whose contents changed during the acquisition process. In our experience, this problem is commonly encountered on systems that have 8 gigabytes or more of RAM installed as well as systems that are under heavy load. Unfortunately, systems with less than 8 gigabytes of RAM are increasingly uncommon and servers frequently have from 16 gigabytes of RAM to hundreds of gigabytes. The increasing amount of RAM installed in computer systems means that nearly *all* captures will contain at least some amount of smear. Depending on where the smear occurs, this can result in undesirable results of varying degrees of severity, from memory pages belonging to one process being assigned to another in the view of the memory analysis tool, to corrupted kernel data structures. Unfortunately, memory analysis tools and frameworks have no method to automatically detect smearing as there is only one source of data for address translation - the smeared page tables themselves.

The move to the cloud and the adoption of local cloud computing models has led to an increase in system utilization across servers. As recommended by both the Amazon AWS (Amazon ec2 container serv, 2016) and Google Cloud (Scaling based on cpu or l, 2016) documentation, running systems at 75% load achieves the greatest balance of CPU utilization without overutilization as well as cost savings. An industry-wide shift to every server acquired running at 75% capacity with 16GB+ of RAM is a very different model than the one used to conceive current generation memory acquisition algorithms.

Issues and limitations

With the exception of attempting to acquire memory as quickly as possible, acquisition tools currently do not make any effort to detect or work around smearing. Analysts routinely try to work around smearing by leveraging hypervisor capabilities when access to the hypervisor host is possible. As discussed in Ligh et al. (2014), virtualization technologies such as VMware, HyperV, and VirtualBox provide the ability to acquire guest memory VM from the host. This acquisition can be performed “instantly” by leveraging hypervisor-specific features, such as snapshots and suspended states, to freeze the guest in-place. This prevents smearing from occurring as the guest can no longer make modifications to memory, and the analyst can simply copy the saved memory state from the hypervisor's file system. This approach is not universally usable, however, as obviously not all systems are virtualized. Furthermore, even in virtualized environments the analyst does not always have access to the physical host, such as in Amazon's AWS or Microsoft's Azure. There may also be issues with temporality suspending a running virtual machine as it may affect production performance and stability.

Future directions

The issues created by page smearing necessitate that memory acquisition tools take smearing into account during the acquisition phase. To this end, we propose several methods that may meet this need.

Leveraging virtual machine hardware extensions. Our first proposed method is a modern implementation of the BodySnatcher tool (Schatz, 2007). BodySnatcher attempted to ‘freeze’ the running operating system and load a small, second operating system at runtime. This was performed without the support of hardware-based virtual machine support as that technology was not yet in use.

We envision that a modern approach to BodySnatcher could instead leverage hardware based virtualization to cleanly:

- Insert a new operating system “under” the existing one
- Freeze the existing operating system
- Write memory of the frozen operating system to removable media or the network

Blue Pill (Rutkowska and Tereshkin, 2008) is a famous example of leveraging this technology to implement rootkit functionality, and the same approach could be taken for defensive acquisition purposes. Besides acquisition from native hardware, this approach would also work for virtual machine guests where the analyst did not have host access, as hardware virtualization extensions allow for nesting of virtual machines. The downside is similar to those of leveraging traditional hypervisors in that network connections and other activity would be frozen for the duration of the acquisition. This is usually not an issue for end-user systems, but is often unacceptable for production servers.

Smear-aware acquisition tools. Our second proposed method is for acquisition tools to become aware of changes to the page tables as acquisition is performed. Unfortunately, there is no single place in the kernel that could be monitored for such changes, as applications and kernel drivers continuously allocate and deallocate memory as well as making changes to memory as the acquisition tool runs.