BodySnatcher: Towards Reliable Volatile Memory Acquisition by Software

By

Bradley Schatz

available at www.sciencedirect.com

ScienceDirect

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

Digital Investigation

BodySnatcher: Towards reliable volatile memory acquisition by software

Bradley Schatz

Evimetry, PO Box 6192, Fairfield Gardens, QLD 4103, Australia

Keywords:
Volatile memory forensics
Memory acquisition
Memory imaging
Digital forensics

ABSTRACT

Recently there has been a surge in interest in memory forensics: the acquisition and analysis of the contents of physical memory obtained from live hosts. The emergence of kernel level rootkits, anti-forensics, and the threat of subversion that they pose threatens to undermine the reliability of such memory images and digital evidence in general. In this paper we propose a method of acquiring the contents of volatile memory from arbitrary operating systems in a manner that provides point in time atomic snapshots of the host OS volatile memory. Additionally the method is more resistant to subversion due to its reduced attack surface. Our method is to inject an independent, acquisition specific OS into the potentially subverted host OS kernel, snatching full control of the host's hardware. We describe an implementation of this proposal, which we call BodySnatcher, which has demonstrated proof of concept by acquiring memory from Windows 2000 operating systems.

1. Introduction

In the formative years of computer forensics the focus of the field was largely on storage media acquisition and analysis, the principles of which are now largely accepted. At the forefront of research today is the topic of volatile memory forensics: the acquisition of memory of a running computer (Carrier and Grand, 2004; Garner, 2006; Farmer and Venema, 2004; Boileau, 2006) and the interpretation of meaningful information from memory images (Farmer and Venema, 2004; Schuster, 2006; Petroni et al., 2006; Casey, 2002; DFRWS, 2005; Carvey, 2006).

This research is motivated by two key factors. Firstly, the emergence of kernel level rootkits and subversion techniques such as Biby (2006) and Rutlowska (2007) has led to a gradual erosion of trust in the integrity of the operating system kernel. While in the past malware ran exclusively in the user level domain of the OS, a new breed of malware, the kernel level rootkit, insinuates itself into the kernel of the OS. The phenomenon of kernel level rootkits complicates the practice of digital forensics by introducing uncertainty as to the correct operation of computer systems. Correspondingly the reliability of any digital evidence that the computer might contain is drawn into question. A means of reliably determining whether a computer was ‘‘operating correctly’’ is required.

Secondly, memory forensics promises to provide a wide variety of evidence related to the state and history of the OS at a fine level of granularity. For example cryptographic keys and passwords might be found in memory, enabling access to encrypted drive volumes (Farmer and Venema, 2004; Casey, 2002). Active communications sessions, and running programs might further be identified.

The focus of the research described in this paper is on identifying means of reliably acquiring memory from arbitrary hosts in the potential presence of kernel level rootkits. Current software based approaches, such as George Garner's *dd* (Garner, 2006), the UNIX */dev/mem* device (Farmer and Venema, 2004), lower level device driver based (Ring and Cole, 2004), and kernel crash dumps (MicroSoft, 2007) are vulnerable to kernel rootkit subversion due to their reliance upon the

E-mail address: bradley@evimetry.com
1742-2876/$ – see front matter © 2007 DFRWS. Published by Elsevier Ltd. All rights reserved.
doi:10.1016/j.diin.2007.06.009

correct operation of the potentially compromised host OS. Hardware approaches such as Carrier and Grand (2004) and Boileau (2006) require ahead of time installation of specific hardware, which is infeasible in all but the most forensically prepared environments. Additionally, recent results indicate that hardware based approaches may be subverted at the hardware memory access level (Rutlowska, 2007), drawing into question the effectiveness of such approaches.

In this paper we propose a software based approach for reliably acquiring host memory by snatching complete control of the host hardware from the running OS. The method injects an independent, task specific minimalist OS into the kernel space of the potentially infected running kernel, precisely saves the state of the running OS, then boots the acquisition OS in an independent and restricted subset of the machine's memory. The acquisition OS then employs a small and common subset of the host's hardware as an output channel for dumping an image of the physical memory of the host.

We present proof of concept of the proposal by describing a prototype implementation of this scheme, demonstrating memory acquisition of Windows 2000 hosts. The prototype employs a custom version of the Linux kernel as the acquisition OS.

This paper is structured as follows. Firstly background material related to memory acquisition approaches is presented. An idealised model of volatile memory acquisition techniques is then presented in order that the benefits and problems associated with the various approaches may be clearly discussed. Section 4 proposes an improved software based approach to physical memory acquisition, and Section 5 describes a prototype implementation of such an approach. Section 6 describes a pair of experiments evaluating the prototype, and finally the limitations of the approach are described, conclusions made and future work outlined.

2. Background and related work

Until recently, the generation of snapshots, or images of sections of memory remained purely of interest to debugging user and kernel level software. The UNIX system and the Windows platform have long provided facilities for dumping the memory and state of user level programs. Core dumps generated in such a way are typically loaded into a debugger for examination.

Many variants of UNIX, and Windows provide for dumping the contents of the entire physical memory of the system. When configured ahead of time, the Windows system generates crash dumps triggered by a number of events, including inconsistent OS states, "magic" keystrokes, and hardware and software faults.

The goals of crash dumps, however, are somewhat at odds with the robustness of the hardware and software. A software or hardware fault triggers the crash dump procedure, which in turn relies on the correct operation of a substantial body of code and hardware in order to correctly dump. Crash dumps are thus vulnerable both to a wide range of software and hardware errors, but remain pragmatically useful when software faults are limited to non-essential drivers and hardware faults to non-essential hardware.

Most UNIX variants provide for crash dumps by running some code from RAM. For example, on LINUX, approaches such as kdump (Goyal et al., 2005) preload a separate spare crash dump specific kernel in order that a clean uncorrupted kernel might make the crash dump, theoretically reducing the effect of software corruption in the host kernel. A number of UNIX hardware platforms, including Sun, AIX and HP anecdotally supported memory dumping by switching control to the firmware residing in the ROM of the machine, by the use of special key sequences or hardware switches (Drake and Brown, 1995).

Employing crash dumps as a means of acquisition in a forensic context is complicated by the potential presence of kernel rootkits. The Windows crash dump facility for example provides hooks which might be employed by the kernel rootkit for subversive purposes. Similarly, hooks in the disk subsystem provide further opportunity for subversive behaviour. Finally, all observed crash dump facilities need to be preconfigured to be used as a memory acquisition device.

2.1. *Software acquisition*

Current software based approaches for memory acquisition typically take advantage of operating system services providing user space access to physical memory as a device file. On LINUX this is the character device */dev/mem*, and for Windows 2000, and non 64-bit Windows XP the device \\.*PhysicalMemory*\. An alternate method is the *NTSystemDebugControl* API call (Vidstrom, 2006a; Garner, 2005). Newer versions of Windows prevent user mode access to this device, necessitating accessing physical memory from a custom device driver.

The most popular current method for acquiring memory on Windows hosts is George Garner's Windows port of *dd* (Garner, 2006), which uses the \\.*PhysicalMemory*\ device. In many cases acquiring memory images via this tool works, however, there are a number of problems associated with this approach.

First and foremost, as this approach is run as user level code on the target host, other processes and large parts of the operating system continue to run while the imaging process takes place. Memory will be continually changed by the continued running of other software, leading to the image not being of a particular point in time, rather, the image is a time lapse shot. Secondly, the technique is not robust, with a number of observed problems related to page caching (Vidstrom, 2006b).

Finally, as the technique depends on the target OS, it is subject to kernel rootkit subversion. Bilby (2006) has demonstrated interception and sanitization of memory acquisition by a standard rootkit technique of hooking the System Service Dispatch Table (SSDT) in his DDEFY prototype. Virtual memory subversion techniques such as those employed by Shadow Walker (Sparks and Butler, 2005) could similarly be used by kernel rootkits to hide from acquisition.

The final current option for capturing memory contents by software is to use the hibernation functionality present in most modern operating systems. Windows for example hibernates by saving a subset of state of the system, including the contents of RAM to a hibernation file on the primary OS filesystem. Such facilities depend on hardware and BIOS support, and are typically found most widely deployed on laptops, server hardware, however, does not typically support this facility. Finally, as with crash dumping, the process is vulnerable to rootkit subversion, as drivers are notified when hibernation begins.

2.2. *Hardware acquisition*

Hardware acquisition methods have been proposed as an alternative method of dumping host memory. Such methods rely on accessing memory through the use of Direct Memory Access (DMA) via a hardware bus. Carrier and Grand proposed and described a prototype PCI card (Carrier and Grand, 2004) which disables the CPU then uses DMA to access host memory, subsequently dumping the memory contents to a connected device. This approach, while precise and atomic, depends on the device being added to the host computer ahead of time. Similar techniques have been demonstrated employing the bus mastering DMA capabilities of the Firewire protocol (Boileau, 2006; Becher et al., 2005). Firewire, however, is not widely deployed, and as such presents problems for reliably imaging memory of arbitrary hosts. The work of Rutlowska (2007) additionally highlights potential new problems with hardware based approaches related to Memory Mapped IO (MMIO) features of emerging chipsets. These features may be used to present a different view of physical memory to hardware devices than is seen by the CPU.

Virtual machines such as VMWare provide a means of taking a snapshot of the state of the virtual machine, which includes the CPU, virtual memory, and the hard disks. The contents of memory are stored in a file. Such memory snapshots are proving useful in research targeted in building analysis tools, and would be acceptable as images where the virtual machine alone is the subject of imaging. However, this technique does not address imaging of the host machine.

3. An idealised model of volatile memory acquisition

The function and traits of an ideal acquisition method for volatile memory are discussed below in order that the different approaches to memory acquisition might be compared.

The function of an ideal memory acquisition method is simple: the volatile memory of a host is sequentially accessed, and its contents are copied out an output channel to a storage medium for preservation. The conventional imperatives of digital forensics dictate that this copy of the memory (or memory image) is a precise copy the original host's memory. The ideal memory acquisition method must be available, working on arbitrary computers (or devices), and additionally must be reliable, either producing a trustworthy result or none at all.

3.1. *Fidelity*

This imperative for high fidelity imaging is most problematic when considering the implementation of memory acquisition methods. Hardware based approaches promise precise, even exact fidelity, through the use of an external and trusted hardware channel, and exclusive access to the memory of the host. Software based acquisition approaches must necessarily relax the requirement of fidelity, by nature of the changes in state which are inevitable in running software on the same host as the target physical memory. The fidelity of images produced in this way can be measured against the following factors: snapshot atomicity, host OS memory integrity, and unallocated memory integrity.

3.1.1. *Snapshot atomicity*

Current software based approaches, by nature of a reliance on the continued normal operation of the computer, are problematic when it comes to the fidelity of the resulting image. The process of accessing the memory, copying it, and then writing it out an IO channel, while the OS and applications continue to run, results in the memory image being imprecise, and not attributable to a specific point in time. Rather, such a memory image is more akin to a time lapse photograph rather than a point in time snapshot. Hardware methods, on the other hand, promise atomic snapshots by nature of shutting down all CPUs for the period of the capture.

3.1.2. *Host OS memory integrity*

Given that any software based acquisition procedure must run on the same hardware as the initial host OS, modification of memory will be inevitable. However, the ideal software memory acquisition process will only cause modifications of the host OS memory as a secondary effect of the actions of the host OS. For example, the Windows based *\\.\PhysicalMemory\ dd* method does not in and of itself modify any host OS memory or data structures; however, by nature of it employing host OS services, the host OS's memory undergoes numerous changes as a direct result of the imaging processes action.

3.1.3. *Unallocated memory integrity*

In media forensics, investigation of unallocated space such as slack space is of considerable interest in identifying information from earlier points in the media's lifetime. Similar types of information lie latent within unallocated memory maintained by the host OS. At odds with this is, however, the strategies used by the OS in caching, as the unallocated memory pages are often used as caching for block device access. The ideal software imaging method should minimise its impact on unallocated memory.

3.2. *Reliability*

Both hardware and software based approaches have been shown to be vulnerable to subversion, the result of which could be denial of service, or data hiding (which correspondingly impacts the fidelity of an image). As highlighted earlier, recent results show that hardware based approaches may be subverted at the hardware memory access level (Rutlowska, 2007). Software based methods are vulnerable to subversion by hooking at a myriad of vectors leading to the observation that "whoever hooks lowest wins".

3.3. *Availability*

The absence of widespread deployment of hardware devices supporting this method of acquisition precludes general adoption of the method, with the *dd* method (anecdotally) currently being the most widely used method of imaging memory.

4. An improved software based acquisition method

In light of the problems observed with current hardware and software acquisition methods, improved methods of memory acquisition are needed. In this section we propose a pragmatic approach to software based memory acquisition. The approach holds the potential of offering the same availability as existing software based approaches, while increasing the fidelity of produced images and reliability of the imaging process.

The fidelity and reliability problems of the existing software based approaches stem from their reliance upon the existing host operating system as a channel by which the memory image may be output. For this reason, we conclude that the reliability of the imaging process may only be assured by an imaging method which operates without resorting to utilizing potentially tainted host operating system code.

We propose the use of specially crafted, light-weight, acquisition operating systems as a solution to these problems. Our proposal is to depose the suspect host OS by snatching all control of the host hardware with a second, acquisition specific OS (we call this an acquisition OS).

Such an OS is a specially modified version of a general purpose OS, specially targeted towards memory acquisition. Acquisition specific functions it performs include the following:

1. the host OS is halted, so that it is in a deterministic state;
2. the host OS memory pages are quarantined in order to preserve the memory pages of the host OS;
3. an output device (from a minimal set) is identified and initialized; and
4. a copy of the physical memory is written to the output device.

Loading the acquisition OS, halting the running kernel and switching control over to the acquisition OS are handled in this case by a host OS specific kernel driver.[1] The driver must necessarily still depend upon the host OS for loading and initially executing the acquisition OS. This approach does, however, have a substantially smaller attack surface due to its lack of reliance on the host OS for actual imaging operation, and as such is less prone to subversion.

The potential benefits of the approach are now identified by employing the criteria identified in Section 3.

Such a method assures that an image is atomic with respect to the host OS, by nature of the pausing of the host OS in a deterministic state. By utilizing only the memory handed it by the host OS, the acquisition OS ensures host OS memory integrity. The method, like all software based approaches, does, however, continue to share the unavoidable problem of impact on the integrity of the unallocated memory of the system. The approach has the potential to be more reliable by nature of having a smaller attack surface than other (host OS dependent) software based approaches.

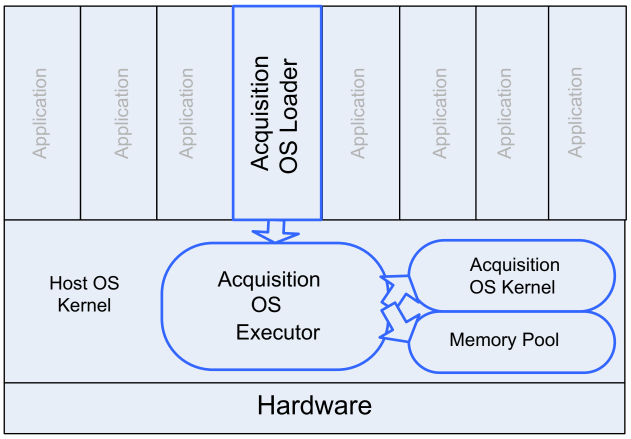

Fig. 1 – Loading sequence prior to deposing the host OS.

5. BodySnatcher: a proof of concept implementation

In this section we describe a prototype implementation of the software acquisition method proposed above. Our prototype is designed for acquisition of volatile memory from Windows 2000 and above machines, running on the i386 platform.

The prototype, which we call BodySnatcher,[2] consists of three parts: the acquisition OS, acquisition OS loader, and acquisition OS executor. All three of these parts are derived from the coLinux project (Aloni, 2004), which is a virtualisation solution enabling running LINUX atop of a Windows OS.

The following sections describe in detail the operation of the BodySnatcher prototype by examining the components of the system (shown in Fig. 1), and key processes involved.

5.1. *Acquisition OS loader*

The acquisition OS loader is a user space application which is responsible for loading both the acquisition OS and acquisition OS executor into memory, then executing the acquisition OS executor in kernel mode with full hardware access privileges. This requires executing the acquisition OS loader using an account with driver loading privileges.

5.2. *Acquisition OS executor*

The acquisition OS loader is implemented as a Windows kernel level device driver, which, running with ring-0 privileges, performs the following tasks:

[1] We expect that other vectors, such as kernel vulnerabilities, might be employed to have the acquisition OS loader executed with full privileges on the CPU.

[2] In reference to the movie "Invasion of the BodySnatchers".

1. Employs the Windows driver API to allocate free pages for the acquisition OS memory pool, kernel and initial ramdisk.
2. The acquisition OS kernel and OS initial ramdisk are then loaded into the memory pool from user space.
3. The driver sets up a virtual address space for the acquisition OS kernel, a trampoline virtual address space for switching between operating systems, and initializes the virtual memory system employed in the acquisition kernel in order to constrain memory accesses only within the pages allocated to the acquisition OS. This is discussed further in Section 5.3.
4. Finally, the loader saves the state of the host OS, and then jumps into a modified boot routine of the acquisition OS kernel.

The current implementation only addresses acquisition on a single CPU machine. Halting the host OS is achieved implicitly when execution passes to Phase 4 above. The feasibility of halting other threads of the OS when running on other CPU cores in SMP and similar configurations will require additional investigation.

5.3. *Memory*

The memory assigned for the acquisition OS to operate within (the acquisition OS memory pool) is allocated from the pool of free pages maintained by the host OS. The acquisition kernel is constrained to operate within this set of pages by means of an additional memory management abstraction interposed between virtual and physical memory: a pseudo-physical-to-physical (PP2P) memory mapping.

This technique presents the acquisition OS kernel with a virtualised physical memory model, constraining the kernel's operation to limit its use of physical memory to only that allocated by the acquisition OS executor. This technique is used by both coLinux (Aloni, 2004) and XEN (Barham et al., 2003) for virtualising memory regions.

The only exceptions to this are the mapping of PCI BIOS regions into the acquisition kernel's memory space, and the custom */dev/mem* implementation. PCI BIOS regions must be mapped into the acquisition OS's address space in order that PCI enumeration, and consequently hardware discovery, might take place. Outside of the acquisition OS memory pool, access to the full range of host physical memory is provided by a custom of */dev/mem* which bypasses the PP2P memory model.

5.4. *Context switch to acquisition OS*

Switching program flow from the host OS kernel to the acquisition OS kernel is complicated on the i386 platform by a lack of an atomic instruction which changes both the Instruction Pointer and Virtual Address space mapping at the same time. This is not typically a problem when running only one OS on a CPU (or CPUs) as, for at least LINUX and Windows on the i386 architecture, the kernel is mapped into the virtual address space of every user process at exactly the same place. Consequently, context switches from user to kernel remain within the same virtual address space. Switches between virtual address spaces occur within the context of the kernel, and only involve changing the virtual address space via the top page directory table pointer (CR3); this implies that the code that performs this change must be mapped into both virtual address spaces at exactly the same virtual address.

Differences between the virtual memory layout of Windows and LINUX mean that it is problematic to load the code necessary for the switch into the same address at both locations. The solution employed by this approach is inherited from coLinux. Position independent code which manages the context switch is mapped into an intermediate virtual address space in two places. This code, along with state information from the host OS, is stored in a ‘‘passage page’’, which is a variation on the traditional approaches of trampolines and continuations.

Referring to Fig. 2, the host OS switches virtual address spaces using the CR3 register, and control lands in the passage page mapping with the equivalent virtual address range. The passage page code then switches EIP to the other mapping of the passage page, which subsequently switches CR3 again to switch into the acquisition OS. Finally, execution jumps to the boot entry point of the acquisition OS.

5.5. *Acquisition OS*

The acquisition OS is a custom version of the coLinux LINUX 2.6.11 kernel with a number of BodySnatcher specific modifications. Beyond the */dev/mem* modifications detailed above, all coLinux virtual hardware is removed, and the kernel pared back to contain only code relevant to acquisition with a constrained set of hardware device. Key subsystems enabled are memory management, interrupt handling, timers, PCI enumeration and serial IO.

Fig. 3 depicts the structure of the running acquisition OS during acquisition. Memory outside that allocated to the

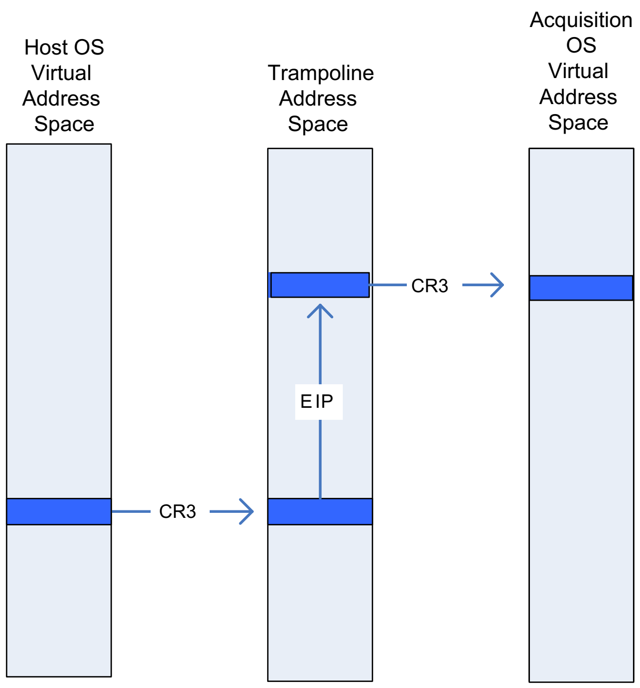
Host OS Virtual Address Space
Trampoline Address Space
Acquisition OS Virtual Address Space
CR3
EIP
CR3

Fig. 2 – Inter-OS context switch involves employing a trampoline virtual address space.

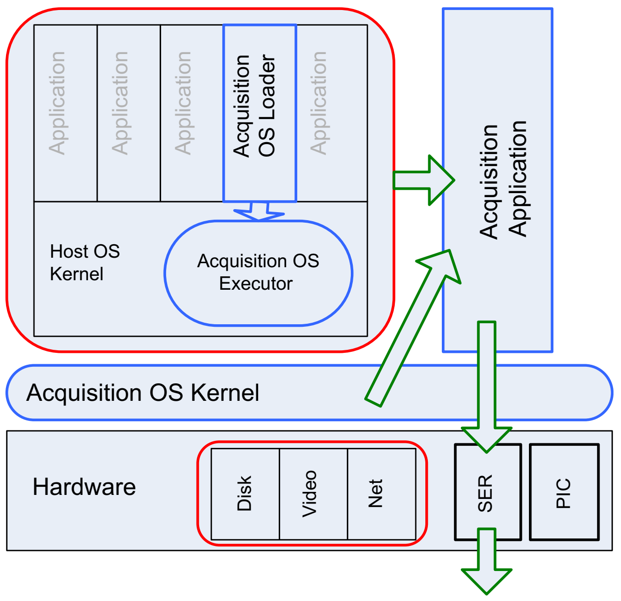

Fig. 3 – Structure of system at acquisition time, demarking immutable objects and data flow.

acquisition OS is effectively immutable, quarantined away from the operation of the acquisition OS, which is represented in the figure as the host OS surrounded by the red rounded box. (For interpretation of the references to colour in Fig. 3 the reader is referred to the web version of this article.)

5.5.1. *Acquired image IO channel*

The output channel employed in this prototype was for simplicity chosen to be a serial port. The kernel ignores all IO related hardware but the serial port, the Programmable Interrupt Controller (PIC), and PCI hardware.

5.5.2. *Other IO devices*

Drivers for no further hardware devices are included in the kernel in order to prevent potential state changing interactions with the host hardware. This is in order to prevent potential evidence spoliation arguments.

Existing IO traffic generated for example by running network cards is silently dropped by the interrupt handlers inside the acquisition OS kernel. Ongoing DMA transactions may modify small driver allocated portions of the host OS memory.

5.6. *Acquisition application*

The process of imaging is simply a shell script, which runs during the usual UNIX init process. The script employs *dd* to run against the specially modified version of */dev/mem*, piping the uuencoded output out the serial connection.

5.7. *Host storage integrity*

Methods for ensuring the integrity of storage media are well defined and understood when it comes to regular hardware preservation. In line with the principle of preserving the integrity of all storage media, the acquisition kernel currently interacts with no storage related hardware.

6. Test procedures and results

In order to validate the effectiveness of the acquisition approach, the following experiments were designed along similar lines to Carrier and Grand (2004) and Petroni et al. (2006).

6.1. *Experiment 1*

A VMWare 5 based virtual host with 128 MB RAM[3] and a virtual serial port was configured to run Windows 2000 SP4. Inspection with windows task manager indicated that around 76 MB of RAM was consumed by the OS, with no significant third party applications running. The acquisition OS was configured to use 32 MB of the host machine's memory. The acquisition kernel was 2.6 MB in size and the initial ramdisk approximately 1 MB in size. The virtual serial port was connected via a pipe to a console session, running the UNIX script command to save the text content of the session.

Five images were acquired:

1. An image of the VM was taken using the VMWare snapshot feature. This image is referred to below as *vmem*.
2. An image of memory was taken to a removable USB device using Garner's *dd*. This is referred to below as *dd*.
3. An image was taken using BodySnatcher immediately after the Garner *dd* image was taken. This is referred to below as *bsdd(1)* (short for BodySnatcher *dd*).
4. The virtual machine was reverted to the system state immediately after taking the snapshot in (1) above. An image was acquired using BodySnatcher at this point. This is referred to below as *bsdd(2)*.
5. The BodySnatcher OS instance was left to run on the host for 8 h, and a second image was taken using this acquisition method. This is referred to in the results table below as *bsdd(3)*.

In order to quantify the extent to which the method caused modifications in the memory of the host machine, the images were compared to determine the number of differing pages in each of the memory images. The results of this comparison are presented in Table 1.

Comparing the two images acquired immediately after the system state snapshot was taken, *dd* and *bsdd(2)*, we see that the host OS dependent *dd* method induced a 43% change in the physical memory of the host, while the BodySnatcher method induced a lesser change of 35%.

It is not possible to distinguish between host OS memory and unallocated memory using the current generation of memory analysis tools, so we are unable to identify the extent to which unallocated memory has been modified.

In order to validate that the BodySnatcher imaging procedure was producing an accurate copy of the host OS memory, the image was analysed using Schuster's *ptfinder* (Schuster, 2006). When run over all images, *ptfinder* (which analyses

[3] This small memory size was adopted for efficiency reasons, both to prevent load on the host machine whose memory resources were scant, and for speed of acquisition. At a serial IO speed of 115 kbps and RAM size of 128 M acquisition takes 45 min.

Table 1 – Number of pages same and different between acquisition methods

	Compared images	Pages same	Pages different	Δ (%)
t_0–t_1	*vmem/dd*	18,536	14,231	43
t_0–t_3	*vmem/bsdd(2)*	21,307	11,461	35
t_1–t_2	*dd/bsdd(1)*	18,905	13,862	42
t_0–t_2	*vmem/bsdd(1)*	15,348	17,420	55
t_3–t_4	*bsdd(2)/bsdd(3)*	32,473	295	0.9

Table 2 – Number of 32 bit words same and different between acquisition methods

	Compared images	Words same ($\times 10^{-3}$)	Words different ($\times 10^{-3}$)	Δ (%)
t_0–t_1	*vmem/dd*	22,206	11,347	34
t_0–t_3	*vmem/bsdd(2)*	25,261	8293	25
t_1–t_2	*dd/bsdd(1)*	23,042	10,511	31
t_0–t_2	*vmem/bsdd(1)*	19,944	13,611	40
t_3–t_4	*bsdd(2)/bsdd(3)*	33,360	194	0.58

Windows memory images to identify processes and threads) identifies processes and threads in all of the images. These processes are consistent, with exceptions based on the introduction of processes and threads associated with the imaging process at hand.

In order to validate that the acquisition OS was not modifying any host OS memory beyond its quarantined allocation, the image *bsdd(3)*, which was taken after the host had been running for around 8 h, was compared with the prior BodySnatcher image *bsdd(2)*. Only a 0.9% change had taken place in memory pages, and on comparison with the memory map of the acquisition OS, all changes occurred within the acquisition OS memory pool.

Comparison of the differences between the Garner *dd* generated image with the BodySnatcher images generated immediately after shows around 2400 more changed pages (13,862 vs.11,461) than is at first glance expected, based on the *vmem/bsdd(2)* comparison, as a result of the application of the BodySnatcher method. Further investigation is required to explain this result.

The number of common 32 bit words was additionally compared; the results are presented in Table 2. These results show similar effects on the proportions of change as was observed with comparing blocks of pages size.

On examination of the memory usage of the acquisition OS, which was accessed via console over the serial port, it was observed that of the 32 MB configured for the acquisition OS to operate within, only 18 MB was used.

6.2. *Experiment 2*

Following observations that the amount of memory dirtied by each imaging process was proportionally high compared with the total RAM of the machine, a second experiment was performed. The same VM was then reconfigured with 512 MB of RAM, and a number of applications loaded into memory in order to more closely resemble real world conditions. Inspection with windows task manager indicated that around 220 MB of RAM was consumed by the OS and applications.[4] The acquisition kernel was a later version of BodySnatcher which was slightly leaner at 2.5 MB in size.

In this experiment only three images were taken:

1. All applications were loaded into memory and a snapshot was taken with the virtual machine. This is referred to below as *vmem*.
2. The VM was reverted to the snapshot, a USB disk mounted, and Garner's *dd* used to image the PhysicalMemory to the USB disk. This is referred to below as *dd*.
3. The VM was then imaged using BodySnatcher. This image is referred to below as *bsdd(2)*.
4. The VM was again reverted to the snapshot, and an image acquired using BodySnatcher. This is referred to below as *bsdd*.

The results of comparing page sized blocks between images are presented in Table 3.

Comparing the number of pages different for *bsdd* compared with *vmem* in Tables 1 and 3, we see that they are both around 11,000 pages (42 MB) different. Further investigation is required in order to identify the cause of differences in the amount of memory change between the two, however, the difference which is 1.7 MB.

While the memory delta caused by the BodySnatcher acquisition stays approximately the same, regardless of host RAM size, the memory delta of Garner's *dd* approach increased almost linearly with the size of the host's RAM. We speculate that this is related both to the caching of the acquired image by the OS block IO layer, and to a lesser extent, continued running of OS and user applications. Further work is required to identify the cause of these.

Comparison of the changes introduced by BodySnatcher after an acquisition by Garner's *dd* again produce a larger number of changes than are expected as was observed in Section 6.1.

7. Known limitations

This approach has many limitations, all of which are tied to the low level nature of the approach. The acquisition OS approach involves far more complexity than existing software based approaches as it involves snatching control of the host entirely away from the existing host OS. This requires an entire new OS in order to operate the host hardware.

Table 3 – Number of changes introduced by acquisition methods under loaded memory conditions

Compared images	Pages same	Pages different	Δ (%)
vmem/dd	70,886	60,185	46
vmem/bsdd	120,045	11,026	8.4
dd/bsdd(2)	73,115	57,956	44

[4] Running applications included all of both the Office 2003, and OpenOffice 2.1 suites and the Eclipse Java IDE.

The most prominent problem with the prototype lies in its highly constrained nature. In order to prove the concept; it has only been validated with Windows 2000 on single CPU VMWare host, the only output channel supported is serial IO (which is increasingly less common, and moreover slow). On attempting to run the same instance of BodySnatcher on a Windows XP virtual machine, the acquisition OS appeared to not fully initialize; instead going into a state where it was consuming 100% CPU for no visible effect. Attempting to run the prototype on a Windows 2000 based workstation on real hardware similarly failed. We suspect that this may be related to our currently only supporting the use of the 'legacy' 8259A Programmable Interrupt Controller, and not newer APICS (which are more common) or that we have not implemented resetting of memory caching functionality.

Creating a generally usable implementation of this method would additionally require addressing graceful shutdown of other processors in SMP systems (which are now becoming mainstream), high memory extensions such as PAE, 64 bit processors such as AMD/EMT64, and newer OS's such as XP, 2003 and Vista. The new policies of Microsoft with respect to loading drivers in Vista will need to be considered in this context. Porting this approach to run on a LINUX based host OS will be trivial by nature of the coLinux underpinnings of the approach, however, how this would fare on other OS's and architectures remains an open question.

The current usage of the serial port as the IO device simplifies the acquisition OS as the IO protocol is extremely simple. Newer IO devices which use DMA require additional changes to the acquisition OS kernel in order to work with the quarantined memory scheme in use. We have avoided IO devices such as network and disk for reasons of simplicity (due to the plethora of network drivers) and integrity of host storage.

The approach still remains vulnerable to subversion via a number of vectors. Firstly, hardware virtualisation features of the newer generation of CPUs might be employed by a rootkit to prevent the acquisition OS from gaining full control of the host OS hardware.

Secondly, the necessity of relying on the host OS for loading the acquisition OS driver and acquisition OS means that at this point in the lifecycle of the acquisition method, the process is vulnerable to attack. Malware would, however, need to distinguish BodySnatcher from otherwise legitimate and necessary kernel driver related operation, which somewhat turns the tables on the usual cat and mouse game of intrusion detection. In this case it is the necessity of malware to detect software which threatens it, and the role of the acquisition tool to hide its intent as much as possible.

8. Conclusions and future work

This paper describes an abstract model by which volatile memory forensics acquisition methods might be distinguished and compared, proposed an improved method of software based memory acquisition, described a prototype implementation of the approach, and evaluated the effectiveness of the prototype.

Such a method has potential to improve upon existing software based approaches in that it provides atomic snapshots of host OS memory at a particular point in time, and is more resistant to subversion by nature of its smaller attack surface. Additionally, we have demonstrated initial results that indicate that the approach has a lower impact on unallocated memory in the host, an improvement on the *\\.\PhysicalMemory\ dd* approach.

As it currently stands, the approach has a similar effect on the resulting hardware state of the hardware host as the recommended and archaic "pull the plug" style of digital crime scene preservation. Currently active IO transactions will be interrupted and dropped by pulling the plug; similarly, transactions in progress in the host kernel will at the time of switch to the acquisition kernel be abandoned.

Next steps will involve widening the availability of the approach by supporting a wider range of OS's and hardware configurations. Secondly, we are currently working on utilizing USB as a high speed and ubiquitous output channel, due to the small number of drivers required to support it.

Future work could explore the potential of resumption of the host OS post acquisition. This is theoretically possible depending on how difficult it will be to save and later resume the state of the hardware devices which are employed by the acquisition OS kernel.

Acknowledgements

The author would like to thank Dan Aloni for both the coLinux version of LINUX, and some helpful feedback. Thanks to the reviewers for their detailed and helpful criticism and suggestions.

REFERENCES

Aloni D. Cooperative Linux. In: Linux symposium, Ottawa, CA, 2004.

Boileau A. Hit by a bus: physical access attacks with Firewire [cited April 2007]. Available from: <http://www.security-assessment.com/files/presentations/ab_firewire_rux2k6-final.pdf>; 2006.

Bilby D. Low down and dirty: anti-forensic rootkits [cited May 2007]. Available from: <https://www.blackhat.com/presentations/bh-jp-06/BH-JP-06-Bilby-up.pdf>; 2006.

Becher M, Dornseif M, Klein CN. FireWire: all your memory are belong to us [cited April 2007]. Available from: <http://cansecwest.com/core05/2005-firewire-cansecwest.pdf>; 2005.

Barham P, Dragovic B, Fraser K, Hand, S, Harris T, et al. Xen and the art of virtualization. In: ACM symposium on operating systems principles, Bolton Landing, NY; 2003.

Carrier BD, Grand J. A hardware-based memory acquisition procedure for digital investigations. J Digit Investig 2004;1(1).

Casey E. Practical approaches to recovering encrypted digital evidence. Int J Digit Evid 2002;1(3).

Carvey H. lsproc released [cited April 2007]. Available from: <http://windowsir.blogspot.com/2006/04/lsproc-released.html>; 2006.

DFRWS. Memory analysis challenge [cited April 2007]. Available from: <http://www.dfrws.org/2005/challenge/index.html>; 2005.

Drake C, Brown K. PANIC! UNIX system crash dump analysis handbook. Pearson Education; 1995.

Farmer D, Venema W. Forensic discovery. Addison Wesley Professional; 2004.

Garner GM. Forensic acquisition utilities [cited April 2007]. Available from: <http://users.erols.com/gmgarner/forensics>; 2006.

Goyal V, Biederman EW, Nellitheertha H. Kdump, a kexec-based kernel crash dumping mechanism. In: Linux symposium, Ottowa, CA, 2005.

Garner GM. Overview of kntlist analysis tool [cited April 2007]. Available from: <http://www.dfrws.org/2005/challenge/kntlist.html>; 2005.

MicroSoft. Windows feature lets you generate a memory dump file by using the keyboard [cited April 2007]. Available from: <http://support.microsoft.com/?kbid=244139; 2007.

Petroni NL, et al. FATKit: a framework for the extraction and analysis of digital forensic data from volatile system memory. Digit Investig 2006;3(4).

Rutlowska J. Beyond the CPU: cheating hardware based RAM forensics [cited April 2007]. Available from: <http://invisiblethings.org/papers/cheating-hardware-memory-acquisition-updated.ppt>; 2007.

Ring S, Cole E. Volatile memory computer forensics to detect kernel level compromise. In: Lecture notes in Computer Science; 2004. p. 158–70.

Schuster A. Searching for processes and threads in Microsoft Windows memory dumps. In: Digital forensics workshop (DFRWS), 2006.

Sparks S, Butler J. Shadow walker: raising the bar for Windows rootkit detection [cited 23 May 2007]. Available from: <http://www.phrack.org/archives/63/p63-0x08_Raising_The_Bar_For_Windows_Rootkit_Detection.txt>; 2005.

Vidstrom A. Memory dumping with NTSystemDebugControl [cited April 2007]. Available from: <http://ntsecurity.nu/onmymind/2007/2007-02-04.html>; 2006a.

Vidstrom A. Forensic memory dumping intricacies – PhysicalMemory, DD, and caching issues [cited April 2007]. Available from: <http://ntsecurity.nu/onmymind/2006/2006-06-01.html>; 2006b.

Bradley Schatz holds a B.Sc. in computer science from the University of Queensland. His Ph.D dissertation, the subject of which is digital evidence representation, is currently under examination. Formerly researching at the Information Security Institute at the Queensland University of Technology, his main research interests are digital forensics, knowledge representation and operating systems. He is the founder of the digital forensics firm Evimetry.