Persistent Systems Techniques in Forensic Acquisition of Memory

Ewa Huebner[1], Derek Bem[1], Frans Henskens[2] and Mark Wallis[2]
[1]School of Computing and Mathematics, University of Western Sydney
e.huebner, d.bem @scm.uws.edu.au
[2]School of Electrical Engineering and Computer Science, University of Newcastle
Frans.Henskens, Mark.Wallis @newcastle.edu.au

Abstract

In this paper we discuss how operating system design and implementation influences the methodology for computer forensics investigations, with the focus on forensic acquisition of memory. In theory the operating system could support such investigations both in terms of tools for analysis of data and by making the system data readily accessible for analysis. Conventional operating systems such as Windows and UNIX derivatives offer some memory-related tools that are geared towards the analysis of system crashes, rather than forensic investigations. In this paper we demonstrate how techniques developed for persistent operating systems, where lifetime of data is independent of the method of its creation and storage, could support computer forensics investigations delivering higher efficiency and accuracy. It is proposed that some of the features offered by persistent systems could be built into conventional operating systems to make illicit activities easier to identify and analyse. We further propose a new technique for forensically sound acquisition of memory based on the persistence paradigm.

Keywords

Persistent Operating Systems, Computer Forensics, Memory Forensics, Memory Acquisition

1. Introduction

Computer forensics has emerged as a distinct discipline of knowledge in response to the increasing occurrence of computer involvement in criminal activities, both as a tool of crime and as an object of crime. The first generation of computer forensics dealt mostly with the capture and analysis of data on permanent storage devices, predominantly hard disks. The standard procedure [26] was to turn off the system under investigation, and transport it to a laboratory, where hard disks were isolated and copied bit by bit using dedicated forensic software. This procedure is still largely adhered to by forensic investigators in many countries [1]. For the evidence derived from the analysis of the disk's content to be valid in a court of law, the disks themselves were protected against any modifications by using hardware write blocking devices, and all subsequent analysis was conducted using the copied images. This approach was successful while the volume of data analysed in this manner was manageable, and while criminals were relatively unsophisticated with regards to hiding evidence of their actions.

With the increasing volume of storage in computers, and with increased knowledge on the part of criminals, conventional methodologies are no longer adequate. Additionally, the growing use of networks such as the Internet, coupled with the increased use of technologies such as Internet virtual storage [2], has expanded the scope of possible locations of data of interest to forensic examiners.

Coupled with the conventional technique of shutting down computer systems in preparation for transport is the loss of all volatile data in the system, i.e. contents of physical memory or data transmitted over current network connections which may be crucial to the investigation. This is the primary reason behind two new approaches that are attracting recent interest in computer forensics:

- analysis of live systems [31]
- memory forensics (capture and analysis of memory image) [20]

Unlike the conventional approach both of these techniques will potentially change the system under investigation and as such any findings may be problematic as evidence in a court of law. When investigating computer related crime the same basic rules applied as in other, more conventional investigations. The first thing every investigator has to be aware of is Locard's Exchange Principle: *"Anyone or anything entering a crime scene takes something of the scene with them, or leaves something of themselves behind when they depart"* [41]. Nevertheless live system analysis and memory forensics are both being vigorously researched, and with further refinement they may eventually be more generally accepted as forensically sound. This acceptance, to a large extent, will depend on the support for forensic software tools that are provided at the kernel level of the operating system. In this paper we will focus on issues related to acquisition of memory images in a forensically sound way, and we demonstrate how this may be achieved by extending the functionality of mainstream operating systems.

2. Current Issues in Computer Forensics

2.1 Conventional computer forensics

Conventional computer forensics investigations deal with the recovery and analysis of data stored in file systems on permanent storage devices, for example hard disks. This can be achieved by using standard utilities, such as dd [21], which take a bit-by-bit copy of the complete device, including the unallocated areas. The images produced in this way can be analysed in a number of ways. For example, the image can be mounted as a logical device on a laboratory system [4], and the file system examined using standard system utilities. It should be noted that for the sake of portability of data most operating systems support the standard file system formats, so incompatibility is unlikely.

Further the image can be analysed using hex editors and dedicated forensic software tools to detect recently deleted files and data hidden in the file system using various methods [27]. Although analysis of images can be very tedious and time consuming, it is possible to discover everything stored on the device. A major advantage of this conventional methodology is that it can be applied without making any modification to the original data.

Unfortunately conventional analysis is becoming progressively hampered by the easy availability of low-cost high volume storage. The methodology suitable for examining gigabytes of storage is simply too inefficient for terabytes or petabytes of data. An additional problem is virtualization of data storage. Rather than using locally installed hard disks for storage of data it is possible to use virtual disks [22] which may use physical devices located in another, even remote place. The conventional forensic methodology is not suited to analysis of such data, and may even fail to detect its existence.

Conventional computer forensics methods are limited to the analysis of permanent data storage and do not capture any volatile data, including the content of physical memory. Such data may contain clues regarding the recent use of the system that is not available anywhere else, and these clues may be crucial to successful prosecution. The next generation of computer forensics opens the possibility of capturing a more complete system state by means of investigations of the live (running) systems well as the capture and analysis of the content of physical memory.

Currently, live analysis of a system is usually conducted using a statically linked tool kit on an external media [25]. Use of these linked tool kits recognises the possibility that utilities already installed on the system may not be safe, because their behaviour may have been compromised prior to forensic examination. Many command line native utilities are available to facilitate the analysis, in particular on UNIX derivative systems, and they can be used to build a portable forensic tool kit. However, while there exists a large number of third-party tools for Windows [34], the complete Windows documentation is not in the public domain, so the level of confidence may not be absolute (e.g. in a court of law).

2.2 Memory Forensics

Capture and analysis of the content of physical memory, known as memory forensics, is currently an area of intense research and experimentation [17, 18, 42]. An instantaneous capture of local memory contents and the content of the local paging file constitutes a snapshot of the complete state of the system, and may reveal information not available elsewhere. A major factor in such analysis is the fact that no software tool is capable of capturing the image of memory without, by the very act of its own execution, changing the content of memory.

The common tool for obtaining a memory dump is dd [21], but there are also other tools built into computer forensics software products, for example EnCase [24], LiveWire Investigator [3], ProDiscover [5] or X-ways Capture [6]. Since software tools executing on the system under analysis will affect the content of the very memory whose state is being captured, their use is inherently flawed in terms of evidence admissibility.

Some attempts have been made to create hardware devices that copy the contents of memory with DMA transfer. An example is the prototype PCI card, Tribble [19], which must be installed on the system before an investigation is conducted. Since installation of PCI devices is typically not possible on running hosts, the usefulness of this technology appears limited. Some experiments have also been conducted to capture the memory image using the FireWire protocol [10, 14], which supports direct memory access (DMA). No new programs need to be loaded into memory to capture its content using DMA, so this hardware-based approach appears superior in a forensic application.

The passage of time is an issue that has not been satisfactorily addressed by either software or hardware methods of memory acquisition. For modern computers, a dump of the whole memory takes considerable time (in an experiment we conducted it took on average 2.5 seconds to copy the memory of 1 Gbyte to a null device on Suse 10 Linux system with an AMD Athlon 1.4 GHz processor). This is a very long time in the life of a computer system during which executing software can effect many changes to the content of memory. This, coupled with the serial nature of memory traversal and capture, ensures that what is ultimately captured will be representative of dynamic change and possibly an incoherent snapshot of memory contents.

Taking a consistent image of memory can only be achieved with the support of the operating system or hardware, which could effectively freeze any activity until the process of acquisition was completed. No commercial operating system directly offers such functionality. One possible exception is the ability of Windows [40] and Mac OS X [43] to create an image of memory for the purpose of hibernation. While going into hibernation the system writes the image as a file to the root directory of the system disk. It is not clear what specific changes to memory and the file system take place in order to make this possible.

Assuming the capability to obtain a snapshot of memory, the next step in memory forensics, interpretation of content, also presents problems [25]. The structure of memory is very heavily system dependent, and may vary even for different versions of the same system. The issues of portability that govern the structure of file systems do not apply here, and there is no need or reason to make memory structure compatible across systems.

Native operating system tools, such as the UNIX crash dump utility [30], are not adequate for a forensic investigator. With Windows the situation is even more difficult, because no complete documentation of the system internals exists in the public domain so the analysis of tool impact on system data is at best problematic. There are partially successful attempts to create software tools for Windows memory dump analysis [17, 18, 42], but they will always lag behind the currently marketed version of the operating system. All such tools are of limited value in recreating the state of the system, because the memory image they analyse is not guaranteed to be consistent and because the tools' execution mutates the memory image.

In summary, operating systems determine which tools can be used and how effective these tools are in a forensic analysis of the system. Currently dominant operating systems do not offer much in these terms, because they were simply not designed to do so. During the 1980s and 1990s a variety of non-conventional operating systems were proposed, which differ substantially from the accepted standard approach. These were termed orthogonally persistent operating systems [8], and their design and implementation constituted a lively area of research. Although persistent systems, mostly for reasons of performance, never entered the mainstream the persistence techniques may be found in commodity operating systems, for example maintenance of temporary backup files in word processors or the System Restore functionality in Windows XP. In the following section we further examine the features offered by persistence at the operating system level, and examine which may be of value to forensic analysis of current computer systems.

3. Consistency of Memory Image

Methods used for memory acquisition, whether software or hardware based, usually include caveats regarding possible changes to memory during the acquisition process. There is some awareness that if the memory dump happens in parallel with the continued execution of the system, it will not capture a self-consistent image of the dynamic system [20].

No software tool can ensure that no other system activity will take place during the production of a memory dump; on the contrary, system activity is necessary for the tool to execute. In comparison, the procedure for use of the Tribble card assumes that the system will be suspended while DMA transfer is in progress. It has, however, not yet been demonstrated that it is possible for the PCI controller to gain complete control over the CPU to stop it executing program code [19]. Typically DMA transfers are initiated by the CPU, and proceed in parallel with normal CPU activity, so such a ‘stop the CPU’ feature seems unlikely for conventionally built systems. Therefore there is no known methodology which would guarantee that the acquired image of memory is a snapshot of memory content [42], rather it has been described as “memory smear”.

As the memory image is being captured, it is also being changed by other parallel processes which interact with memory, modifying kernel structures and user address spaces, as shown in Figure 1. Self-consistency is violated because of the sequential nature of the memory acquisition process. As a consequence pointers from saved memory may refer to data that has changed before the pointed-to memory is saved, resulting in illegal references or inconsistent analysis of data (see figure 1c and 1d). Also data created later in memory that has already been saved (pointers or objects), may result in dangling references or unreachable objects (see figure 1a and 1b). Additionally there is a possibility that the alleged perpetrator may have a process in execution for the purpose of destroying data in memory in parallel with the memory dump. This could be achieved, for example, by aggressively allocating large areas of memory to the process, which then overwrites it.

For the above reasons, we can consider a memory dump achieved in parallel with system execution as a technique that produces non-self-consistent data, and potentially loses data of interest. While this data may reveal valuable information, it is less useful from a forensic point of view than data that is self-consistent and therefore more readily interpreted. Moreover, because the memory content has been mutated while the computer is in the hands of investigators (rather than representing the state left by the perpetrators), findings based on the analysis of such images could potentially be successfully challenged if presented as evidence in a court of law. The legal profession has serious difficulties with coming to agreement on how to handle and treat computer related evidence even when dealing with less complex issues. Understanding of basic terms when they are used in relation to conventional evidence handling is rarely questioned, while the same terms are often questioned when they are used in relation to computers. For example there is no clear agreement what does it mean to “search” computer data or when is computer data “seized” [33].

Mainstream operating systems provide various tools for capture and analysis of memory, which were developed to facilitate debugging and analysis of system crashes. These tools often underpin the development of forensic tools [39], but for various reasons, including difficulty of usage, they are not directly suitable. An important additional point is that for virtual memory systems, an image of core memory (RAM), even assuming that it is accurate, does not represent the complete state of the system. To truly represent the virtual addressing environment we would also need to include the content of paging or swap areas (be they files or partitions) which belong to the same snapshot as the captured physical memory content.

No mainstream operating system provides a native tool for capturing the complete and consistent state of the system. Orthogonally persistent operating systems, on the other hand, are built with the fundamental underlying assumption that the system can capture its own state, thus providing a continuation point after system restart or recovery from failure. This state is guaranteed to be self-consistent, so for example it is possible to turn the power off, and later restart the system with a minimal loss of state. This feature of persistent systems would be particularly valuable to forensic investigations, because it would allow for complete analysis of the system off site in a trusted laboratory environment. Importantly the stored system state would include no additional data introduced since investigator intervention.

4. Persistence in Operating Systems

The ability to capture a self-consistent memory image, termed stability, is inherent to an orthogonally persistent operating system. One of the earliest implementations of a persistent system was Napier88 [15]. This system used a global checkpoint to capture its own stable state including the image of memory. Napier88 used a so called "stop-the-world" approach to stability in which all user program execution periodically halted while the system state was recorded on non-volatile storage. The Casper system [46] implemented a distributed version of Napier88, and introduced an incremental checkpoint mechanism based on the dynamic recording of dependencies between objects in so called associations. All objects in an association had to be checkpointed together. This scheme supported simultaneous user-level activity in some parts of the store while other parts of the store were stabilised. To maintain self-consistency, all entities in an association had to be stabilised (or rolled back) together in an atomic operation. The gains in user perception of performance resulting from incremental checkpoints were offset by the need to maintain the associations and lack of control over false dependencies.

The stability scheme provided in MONADS/DSM [32] also implemented an incremental approach, which was based on the use of directed dependency graphs (DDGs) to record relationships between entities [29]. The use of DDGs allowed fine-grained analysis of dependencies, significantly decreasing the time taken for each checkpoint or rollback operation by avoiding domino effect propagation. Another persistent operating system, KeyKOS [35], maintained a log of modified objects, and implemented an incremental checkpoint which executed as a background process, moving the objects between the log and stable storage. This resulted in a significant performance penalty, as each object had to be moved twice, first to the log and then to the disk.

Inefficiencies of global checkpointing and/or the inability to completely eliminate false dependencies in incremental checkpointing mechanisms were the motivation for more experimental designs, many of which were proposed and partially implemented in the Grasshopper system [23]. Some of these designs were based on a partitioned store with dependencies tracked using optimistic strategies [36]. Optimistic strategies may result in a substantial rollback of the system to reach a point were all dependencies between objects are captured with the only bound being the initial state of the system. A later design in Grasshopper used fault tolerance methods to increase the stability and resilience of persistent objects [12] by reducing the volatility of main memory. It should be mentioned that persistence of processes, including of the system itself, presents a particularly challenging problem that has not been completely solved by any persistent system [13].

The problem with all of the above, except the global checkpoint "stop-the-world" approach in Napier88, is that the content of memory is still acquired in parallel with system execution of the usual kernel and user level processes. The resultant memory image does not represent a snapshot of any instantaneous system state. Rather, the stable image represents a time continuum, predicated by the understanding that the data is both consistent (i.e. it is devoid of dangling pointers and unreachable objects) and logically self-consistent (i.e. data versions reflect temporal ordering of updates to the data). Thus, unless it was created by an orderly system shutdown, the stable image will almost certainly exclude the most recent mutations to data in the store.

For the purpose of forensic examination, it is perhaps preferable that the captured image represents the equivalent of an instantaneous snapshot of the system's dynamic memory resembling the "stop-the-world" approach used in Napier88. The data captured in this way in combination with data in the paging/swap area records the address spaces of all processes on the system at the time the image was acquired. This would show the content of scheduler data structures, virtual memory page tables, the content of file buffers, the content of other I/O buffers (e.g. network, file buffer cache, kernel random number generator), all processes stack, data and text segments, previous data contained in page frames on the free list, etc. In addition the references and data structures would all be completely consistent from the point of view of interpretation by the operating system. This last observation leads to the possibility of building memory image analysis tools based on code extracted from the operating system implementation.

5. Relevant Features of a Persistent System

An operating system is orthogonally persistent if it fulfils the following requirements:

- The system must support creation and maintenance of persistent objects as a basic abstraction.

- The state of processes must be contained within the persistent objects.
- A protection mechanism for persistent objects must be provided.
- Persistent objects must be stable and resilient, i.e. they must survive system malfunction.

The features relevant to forensic investigation are the ability to capture the state of the system within the persistent objects, and the guarantee of stability and resilience for these objects. An orthogonally persistent operating system has to support these abstractions at all times, a requirement that is unnecessarily strong from the forensic point of view.

The difficulties in finding an appropriate methodology for acquisition and analysis of physical memory are not as significant for a persistent system. The content of memory which is relevant to the system is captured periodically by the internal mechanisms that ensure persistence, and the whole system can be analysed using that preserved state. When the system is restarted from the most recent checkpoint, all stable information will become available again.

The perennial issue in any computer forensics investigation is how to assume possession of the computer system under investigation. One question is when to remove power from the system to enable transfer, either immediately or after collecting some evidence from the running system. Another question is how to do it, either by pulling out the power cord from the system or by shutting it down properly [16]. The conventional wisdom was that cutting off the power is the very first activity to be taken in a crime scene in order to ensure that no data will be modified in any way by the investigator's activities. This attitude is now changing with the realisation of the benefits of live investigations and memory forensics [47]. In the absence of stringent and well understood mechanisms and procedures, the consequence of the decision not to power off the system may lead to a successful challenge to the validity of findings in a court of law.

This dilemma potentially disappears when dealing with a persistent operating system. The persistent system aims to preserve its state under the circumstance of un-ordered shutdown (e.g. sudden loss of power). In practice some of the state may be lost, but it is always possible to restart the system with a loss of no more than single minutes of data mutation. It is significant to note, that after the system was turned off the image of all permanent storage devices (and hence the stable image of memory) can be taken and preserved using a conventional forensic methodology.

Once the persistent system is restarted the investigator returns to the moment shortly before the time the system was powered off. If the investigation is to lead to prosecution then any findings on the live system can be confirmed using the preserved and protected images of the permanent storage, so the problems inherent in the live investigation of commercially available systems do not arise.

6. Forensic Investigation of a Persistent System

Let us consider a hypothetical scenario with an orthogonally persistent operating system being the subject of a forensic investigation. In addition to periodic automatic (global or incremental) checkpoints a persistent system usually provides the ability to checkpoint on demand, which is required to support transaction-based applications. All the investigator needs to do is to optionally request a system checkpoint, turn off the system, and transport it to the laboratory (subject to earlier observations with respect to the changes this checkpoint makes to the system state). To prevent malicious interference, the system would need to support suspension of user-level process activity during this form of system checkpoint. Alternately the investigator would simply turn off the system and rely on the system-created stable state.

In general terms the system checkpoint utility performs the following steps:

- disable interrupts to ensure it will execute without context switches, which essentially precludes the possibility of malicious code executing in parallel with the checkpoint,
- checkpoint the kernel and each user process by copying its stack, data and text segment, and
- checkpoint all kernel data structures in memory.

From the persistence point of view a significant difficulty is in handling those user processes which currently execute in kernel mode, and are in the middle of modifying kernel data structures [13]. Some of the kernel data structures relate to transient objects which cannot be recreated on restart, for example open network connections. This problem is not as acute if the checkpoint utility is used for forensic purposes, and it is therefore not expected that the system will be able to continue seamlessly after restart. It is acceptable that when the previous system state is recreated in laboratory conditions some user processes will crash, as long as the cause of the crash is recognizable.

Once the whole system state is captured, the system can be shut down or even turned off. From that moment conventional forensic methodology could be applied to the system; it could be transferred to a laboratory, where disk images would be obtained and preserved for further analysis. In addition the stable state of the system could be loaded in a virtual environment [11] to recreate as much as possible of the live system, and to analyse its dynamic behaviour. Unless an on-demand checkpoint is performed there may be some small loss of state when the system is turned off, so the conventional approach to memory acquisition and analysis may still be applicable in a persistent system. The importance of this loss of state on the ultimate findings is much reduced, because losses only result from the small amount of time between the last checkpoint and the disconnection.

It is worth noting that the above forensic implementation describes the essence of what was formerly called the “stop the world” approach to checkpointing. This approach was superseded (by incremental checkpoint) in persistent systems because of its detrimental effect on user activity for the duration of the checkpoint operation. Because of the highly specific nature of its purpose, a similar utility to “stop the world” could be implemented as an extension of a conventional operating system, since continued system operation is not required (or indeed desired) in parallel with, or on completion of, the checkpoint operation.

7. Capturing Self-Consist State in a Conventional System

The software tool we propose for capturing a system’s consistent state has to be part of the kernel of the operating system in order to gain access to the lowest levels of control, for example setting the interrupt priority levels and ability to execute privileged instructions. This tool will, on demand, capture the complete virtual memory image comprising physical memory and the content of the paging/swap area, and store the images on an external device.

At system boot a protected area of physical memory is reserved, providing a range of physical and system addresses that is never used during normal operation of the system. Further, this address range is locked in memory, similarly to the non-paged pool in Windows [40]. In this segregated memory, the system loads the required code and reserves space for the data buffers required to achieve a dump of the remainder of the physical memory and the paging/swap area. The code lies dormant until it is activated by an extension to the functionality of the known secure attention sequence of keystrokes (SAS) [44], Ctrl-Alt-Del, designed to minimise the possibility of accidental occurrence.

Any keyboard input generates a high priority hardware interrupt, which triggers a context switch to INT 9 interrupt handler, part of the keyboard device driver belonging to BIOS [28]. For all keys and key combinations, except Ctrl-Alt-Del, this handler converts the character scan code to ASCII, and passes control to the software interrupt handler, INT 16. Because Ctrl-Alt-Del is trapped by the interrupt handler and passed to the operating system, it cannot be intercepted by an application program executing in user mode. It is therefore suitable for implementing various secure system functions. This key combination was originally developed by IBM to provide means for soft reboot on a PC, and since has been utilised in Windows operating systems by Microsoft to provide means of secure login and other sensitive system tasks.

For example in response to SAS the Windows XP (and later) handler starts a utility Winlogon [44]. This utility either displays the login window or if a user is already logged into the system, the Windows Security dialog box with a number of options, including log-out, shutdown and task manager. Some Linux distributions also take advantage of the Ctrl-Alt-Del sequence by trapping the generated interrupt and providing alternate system functionality within the interrupt handler [7].

Addition of another option to the utility invoked by the INT 9 interrupt handler will provide for activation of the memory acquisition code. This code will record the content of memory to an external USB device provided by the forensic investigator – importantly avoiding alteration to any attached storage belonging to the suspect computer system. Inclusion of a generic USB driver in the code will ensure the ability to access the investigator-introduced storage device without reliance on (possibly corrupted) existing system drivers.

Microsoft Windows provides a similar method for performing a memory dump; by pressing the right-CTRL key and the SCROLL LOCK key twice [38]. The effect of this key sequence is that Windows copies all or part of its physical memory into the swap area of the system. There are many restrictions on the usage of this facility, for example it is dependent on the hardware configuration and the size of the swap area. In addition, in its current form, this feature must be activated in the registry before it can be used. Most importantly the key sequence used is not secure and can be intercepted by an application program.

The memory acquisition code is invoked by the system interrupt handler, so it executes after pre-emption of process(es) running at the time of the interrupt. Consequently all information about the pre-empted process(es) is stored in the process control block in preparation for process re-scheduling. Process control blocks form part of the memory image captured by the acquisition code, providing a snapshot of system activity for later analysis. While the acquisition code is executing, interrupts are turned off to ensure the code’s continuous execution and to exclude the possibility of malicious interference.

The acquisition code itself is carefully constructed so that nothing it does changes even a single byte of the memory outside the segregated area. Its sole purpose is to save the contents of the main memory, and of the swap area, to external non-volatile storage. The result is a snapshot of the system state at the time of the jump into the segregated code. Because that jump occurs after pre-emption of the process that was executing at the time of the Ctrl-Alt-Del key combination, the state of this process is also preserved.

With the resultant memory image and the content of the paging/swap area it is possible for tools that have the same level of understanding of the system structures as the original operating system to answer all questions about the system state at the time of the dump. For example it would be possible by querying the scheduler’s ready and wait lists to identify all the running processes and their owners. For each of these processes it would be possible to interpret the content of their text, data and stack, identify open files and network sockets. This constitutes complete information about each process.

After the system state has been captured it is possible to follow two independent paths. The capture code can either: terminate and return the system to its previous activity, or it can shut the system down as its final task. If the system is to be shut down then the swap area which corresponds to the time instant of the memory dump would already exist on the hard disk, and for the purpose of efficiency it would not be necessary to capture it again. On the other hand continuation of system activity after completion of the memory acquisition may be necessary if the investigator decides to perform live system analysis or allow the system to complete some interrupted activities. For example it may be desirable to allow an interrupted network download to complete before shutting down the system. In such case capture of the swap area is needed to ensure consistency.

8. Security considerations

Any software tool, be it an application or a part of the operating system kernel, is vulnerable to attacks, and is only as safe as the security measures used on the system. The same is true for the software tool described in Section 7, which must be protected from malicious modification or deactivation to perform its function. In particular, it should be difficult for an end user to disable the tool, thus hampering or preventing forensic analysis of their computer-based activities. In the following discussion it is assumed that the software tool is always enabled by default and does not require any initial configuration.

Potential attackers can be broadly categorized into two groups, each of which would interact with the operating system in a different way. The first category includes unskilled users, who may not be aware that such a software tool is in place, and as such would not attempt to disable it, and users who are aware of the tool but nevertheless decide to leave it in place. This category of users poses no threat to the presented memory acquisition technique.

The second category of attacker is the professional criminal who knows the software tool is in place and takes actions to disable it.

Attacks against the tool can only come in the form of software that is developed by an experienced systems programmer but may be distributed for use by less-skilled attackers. This approach is a familiar one [45]. Attack against the tool itself can take two forms. An attack may attempt to capture the key sequence or the interrupt by replacing the original INT 9 interrupt handler, or attempt to modify or replace the code that implements the tool, including overwriting the segregated area of memory. Any form of attack relies on an attacker having privileged access to the system in a way that allows them to alter the underlying operating system code.

This access may be gained legitimately by the owner of the system, and in such case there are no software security mechanisms to prevent alterations including installation of a completely different operating system. The access may also be gained by means of root kits, which are installed on the system after a break-in. This is no different from attacks on any other system function, and prevention of such attacks is the domain of system security, which is beyond the scope of this paper.

Some protection may be achieved using hardware support. An example of such hardware protection is incorporated in the Xbox Gaming Console [37]. In this computing device the manufacturer signs the official firmware [9] with a protected key which is hardwired into the central processing unit (CPU). An algorithm that checks the firmware against the hardwired key is also stored in the CPU. Upon system boot, and at regular intervals afterwards, the CPU generates a hash for the stored firmware and compares that hash with the protected key. Given the required CPU-based support, this technique can also be applied to protecting components of the operating system from alteration. After protection has been placed around the keyboard interrupt handler and the software tool, the protection hardware detects any interference at the time of its occurrence rather than at the time of execution of the protected code. This prevents unwanted interference by halting the system as soon as memory tampering is detected. This type of hardware protection is not infallible, and can be disabled by the owner of the system. For example shortly after release of Xbox it was possible for the owner to have the system converted for a modest charge, turning it into a general purpose computer.

9. Conclusions and Future Work

Although much progress has been achieved in the area of system security, no security provisions offer absolute guarantees. Therefore any system can be abused and ultimately become the subject of a forensic investigation. Likewise, any computer system independently of its security arrangements can be used to support non-computer related criminal activities. Current mainstream operating systems offer little support to a forensic investigator, which is understandable because this was never an explicit goal in their design.

The inherent ability of orthogonally persistent operating systems to capture their own state suggests a very useful forensic capability. If conventional operating systems were constructed with that capability (albeit un-used by typical users) it would remove the need for live system investigation as it is now understood, and make memory acquisition and analysis as forensically sound as the conventional forensic methodology in use for hard disks.

This paper showed how the ability to save its own state could be added as a kernel utility to popular conventional operating systems including Windows and UNIX. A prototype of the software tool is under construction for UNIX, using an open source version of this operating system. Further investigation aims to provide the ability to commission the utility as a pseudo device driver that is loadable at boot time.

Memory acquisition is only the first step in forensic memory investigation; the second is interpretation of the acquired memory image. The data which could be captured by the tool presented in this paper represents the complete system state, and the ultimate tool for its interpretation is the operating system itself. Future research will investigate the use of a cut down version of the operating system to underpin building of interpretation and analysis tools.

References

[1] *Guidelines for best Practice in the Forensic Examination of digital Technology*, 2002, http://www.ioce.org/fileadmin/user_upload/2002/ioce_bp_exam_digit_tech.html.

[2] Internet Virtual Storage, (2007). Retrieved 20 November 2006, from http://www.cryer.co.uk/resources/virtualstorage.htm.

[3] LiveWire Investigator, (2007). WetStone Technologies Inc. Retrieved 12 May 2007, from http://www.wetstonetech.com/f/forensictools.html.

[4] Mount Image Pro, (2007). Retrieved 12 February 2007, from http://www.mountimage.com/.

[5] Technology Pathways, LLC, ProDiscover Forensics, (2006). Retrieved 20 October 2006, from http://www.techpathways.com/ProDiscoverDFT.htm.

[6] X-Ways Software Technology AG, X-Ways Software for Forensics, Data Recovery and IT Security (2007). Retrieved 1 March 2007, http://www.winhex.com/.

[7] G. Anderson and C. Karakas, *GNU/Linux Command-Line Tools Summary*, in Linuxtopia, ed., 2006.http://www.linuxtopia.org/online_books/gnu_linux_tools_guide/index.html.

[8] M. P. Atkinson and R. Morrison, *Persistent Systems Architecture*, *The Proceedings of the 3rd International Workshop on Persistent Object Systems*, 1988, pp. 1-22.

[9] P. Barth, J. Mears and A. Green, *Project B (Hacking) Overview*, *Xbox-Linux*, 2002, http://www.xbox-linux.org/docs/projectboverview.html.

[10] M. Becher, M. Dornseif and C. N. Klein, *FireWire all your memory are belong to us*, *CanSecWest*, Vancouver, BC, Canada, 2005.

[11] D. Bem and E. Huebner, *Computer Forensics Analysis in Virtual Environments*, International Journal of Digital Evidence, accepted for publication (2007).

[12] E. Z. Bem and J. Rosenberg, *An Algorithm for Stabilising Multiple Stores*, *The Proceedings of the 9th Workshop on ACM SIGOPS European Workshop*, 2000, pp. 139 - 145.

[13] E. Z. Bem and J. Rosenberg, *An Approach to Implementing Persistent Computations*, *The Proceedings of 9th International Workshop on Persistent Object Systems*, 2000, pp. 189-204.

[14] A. Boileau, *Hit by a Bus: Physical Access Attacks with Firewire*, *Ruxcon*, University of Technology Sydney, Australia, 2006.

[15] A. L. Brown and J. Rosenberg, *Persistent Programming Systems: An Implementation Technique*, in G. S. A. Dearle, S. Zdonik, ed., *4th International Workshop on Persistent Object Systems*, Morgan Kaufmann, 1990, pp. 199-212.

[16] S. Bunting and W. Wei, *EnCase Computer Forensics: The Official EnCE: EnCase Certified Examiner Study Guide*, Wiley Publishing, Indianapolis, IN, 2006.

[17] M. Burdach, Digital forensics of the physical memory, (2005). from http://forensic.seccure.net/.

[18] M. Burdach, Windows Memory Forensic Toolkit, (2007). from http://forensic.seccure.net/.

[19] B. D. Carrier and J. Grand, *A Hardware-Based Memory Acquisition Procedure for Digital Investigations*, Digital Investigations Journal, 1 (2004).

[20] H. Carvey, *Windows Forensic Analysis*, Syngress, Rockland, Massachusetts, U.S.A., 2007.

[21] E. Casey, *Digital Evidence and Computer Crime*, Elsevier Academic Press, London, UK, 2004.

[22] T. Clark, *Storage Virtualisation technologies for Simplifyng Data Storage and Management*, Pearson Education, Upper Saddle River, NJ, 2005.

[23] A. Dearle, R. d. Bona, J. Farrow, F. A. Henskens, A. Lindstroem, J. Rosenberg and F. Vaughan, *Grasshopper: An Orthogonally Persistent Operating System*, Journal of Computing Systems, 7 (1994), pp. 289-312.

[24] EnCase, Guidance Software Development and Training, (2007). Retrieved 9 February 2007, from http://www.guidancesoftware.com/training/.

[25] D. Farmer and W. Venema, *Forensic Discovery*, Addison-Wesley, Upper Saddle River, NJ, 2005.

[26] S. V. Hart, *Forensic Examination of Digital Evidence: A Guide for Law Enforcement*, U.S. Department of Justice Office of Justice Programs, Technical Working Group for Electronic Crime Scene Investigation, April 2004.http://www.ojp.usdoj.gov/nij/pubs-sum/199408.htm.

[27] E. Huebner, D. Bem and C. K. Wee, *Data hiding in the NTFS file system*, Digital Investigation, Volume 3 (Spring 2006).

[28] R. Hyde, *The Art of Assembly Language Programming*, 2000.http://www.arl.wustl.edu/~lockwood/class/cs306/books/artofasm/.

[29] R. Jalili and F. A. Henskens, *Using Directed Graphs to Describe Entity Dependency in Stable Distributed Persistent Stores*, *Proceedings of 28th Hawaii International Conference on System Sciences*, 1995, pp. 665-674.

[30] M. K. Johnson, *Red Hat, Inc.'s Network Console and Crash Dump Facility*, 2007, http://www.redhat.com/support/wpapers/redhat/netdump/.

[31] K. Jones, R. Bejtlich and R. Curtis, *Real Digital Forensics: Computer Security and Incident Response*, Penguin Books Pearson Publishing, 2005.

[32] J. L. Keedy, *Support for Objects in the MONADS Architecture*, *Proceedings of the Third International Workshop on Persistent Object Systems*, Springer, Newcastle, New South Wales, Australia, 1989.

[33] O. S. Kerr, *Searches and Seizures in a Digital World*, 119 Harv. L. Rev. (2005-2006).

[34] W. G. Kruse II and J. G. Heiser, *Computer Forensics: Incident Response Essentials*, Addison Wesley Professional, 2002.

[35] C. R. Landau, *The Checkpoint Mechanism in KeyKOS*, *Proceedings of the 2nd International Workshop on Object Orientantion in Operating Systems*, IEEE, 1992, pp. 86-91.

[36] A. G. Lindstroem, A. Dearle, R. d. Bona, J. M. Farrow, F. A. Henskens and J. Rosenberg, *A Model for User-Level Memory Mangement in a Persistent Distributed Environment*, in G. Gupta, ed., *Proceedings of the 17th Australasian Computer Science Conference*, New Zealand, 1994, pp. 343-354.

[37] Microsoft, *Microsoft Xbox*, http://www.xbox.com/en-US/.

[38] MicroSoft, *Windows feature lets you generate a memory dump file by using the keyboard*, 2007, http://support.microsoft.com/default.aspx/kb/244139/en-us.

[39] J. N. L. Petroni, A. Walters, T. Fraser and W. A. Arbaugh, *FATKit: A framework for the extraction and analysis of digital forensic data from volatile system memory*, Digital Investigation, 3 (2006), pp. 197-210.

[40] M. E. Russinovich, Solomon, D.A. , *Microsoft Windows Internals*, Microsoft Press, Redmond, 2005.

[41] R. Saferstein, *Forensic Science Handbook*, Prentice Hall, 2001.

[42] A. Schuster, *Searching for processes and threads in Microsoft Windows memory dumps*, *The Proceedings of the 6th Annual Digital Forensic Research Workshop* 2006, pp. 10-16.

[43] A. Singh, *Mac OS X Internals: A Systems Approach*, Addison Wesley Professional, 2006.

[44] D. A. Solomon and M. E. Russinovic, *Inside Microsoft Windows 2000, Third Edition*, Microsoft Press, 2000.

[45] L. Spitzner, *Know your Enemy – The Tools and Methodologies of the Script Kiddie*, *Honeynet Project*, 2000, http://www.honeynet.org/papers/enemy/.

[46] F. Vaughan, T. Basso, A. Dearle, C. Marlin and C. Barter, *A Cached Architecture Supporting Persistence*, Computer Systems, 5 (1992), pp. 337-359.

[47] J. Zittrain, *Searches and Seizures in a Networked World*, 119 Harv. L. Rev. F. (2006).