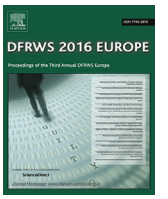
Digital Investigation 16 (2016) S1–S10

Contents lists available at ScienceDirect

Digital Investigation

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



DFRWS 2016 Europe — Proceedings of the Third Annual DFRWS Europe

Evaluating atomicity, and integrity of correct memory acquisition methods

Michael Gruhn*, Felix C. Freiling**

Department of Computer Science, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Martensstr. 3, 91058 Erlangen, Germany

Keywords:
Memory acquisition
Atomicity
Memory forensics
Integrity
Forensic tool testing

A B S T R A C T

With increased use of forensic memory analysis, the soundness of memory acquisition becomes more important. We therefore present a black box analysis technique in which memory contents are constantly changed via our payload application with a traceable access pattern. This way, given the correctness of a memory acquisition procedure, we can evaluate its atomicity and one aspect of integrity as defined by Vömel and Freiling (2012). We evaluated our approach on several memory acquisition techniques represented by 12 memory acquisition tools using a Windows 7 64-bit operating system running on a i5-2400 with 2 GiB RAM. We found user-mode memory acquisition software (ProcDump, Windows Task Manager), which suspend the process during memory acquisition, to provide perfect atomicity and integrity for snapshots of process memory. Cold-boot attacks (memimage, msramdump), virtualization (VirtualBox) and emulation (QEMU) all deliver perfect atomicity and integrity of full physical system memory snapshots. Kernel level software acquisition tools (FTK Imager, DumpIt, win64dd, WinPmem) exhibit memory smear from concurrent system activity reducing their atomicity. There integrity is reduced by running within the imaged memory space, hence overwriting part of the memory contents to be acquired. The least amount of atomicity is exhibited by a DMA attack (inception using IEEE 1394). Further, even if DMA is performed completely in hardware, integrity violations with respect to the point in time of the acquisition let this method appear inferior to all other methods. Our evaluation methodology is generalizable to examine further memory acquisition procedures on other operating systems and platforms.

Introduction

Volatile memory (RAM) is an increasingly valuable source of digital evidence during a forensic investigation. Not only are cryptographic keys for full disk encryption kept in RAM, but also many other pieces of information like the list of running processes and the details of active network connections are kept in RAM and are lost, if the computer would be simply turned off during evidence collection.

There are many ways to acquire volatile memory on standard desktop and server systems today (Vömel and Freiling, 2011). The possibilities range from software-based methods with tools like mdd[1] or WinPMEM,[2] over DMA attacks (Becher et al.) up to cold boot attacks (Halderman et al., 2009). All these methods have their advantages and disadvantages. On the one hand, while software-based methods are very convenient to use, they can be subverted

* Corresponding author.
** Corresponding author.
E-mail addresses: michael.gruhn@cs.fau.de (M. Gruhn), felix.freiling@cs.fau.de (F.C. Freiling).

[1] http://sourceforge.net/projects/mdd/.
[2] http://www.rekall-forensic.com/.

http://dx.doi.org/10.1016/j.diin.2016.01.003

by malware (Stüttgen and Cohen, 2013). On the other hand, DMA and cold boot attacks are often defeated by unfavorable system configurations (BIOS passwords or inactive DMA ports) or technology-immanent problems (Gruhn and Müller, 2013). Overall, these hindrances might produce memory images that are not forensically sound. To what extent this happens, is still rather unclear.

To address this point, Vömel and Freiling (2012) integrated the many different notions of forensic soundness in the literature into three criteria for snapshots of volatile memory: (1) correctness, (2) atomicity and (3) integrity. All three criteria focus on concrete requirements that are motivated from practice:

- A memory snapshot is *correct* if the image contains exactly those values that were stored in memory at the time the snapshot was taken. The degree of correctness is the percentage of memory cells that have been acquired correctly
- The criterion of *atomicity* stipulates that the memory image should not be affected by signs of concurrent activity. It is well known that unatomic snapshots become "fuzzy" (Libster and Kornblum, 2008). The degree of atomicity is the percentage of memory regions that satisfy consistency in this respect.
- A snapshot satisfies a high degree of *integrity* if the impact of a given acquisition approach on a computer's RAM is low. For instance, by loading a software-based imaging utility into memory, specific parts of memory are affected and the degree of system contamination increases (and consequently, integrity decreases).

All three criteria were formally defined and shown to be independent of each other.

With these criteria it was now possible to measure and not only estimate the forensic soundness of snapshot acquisition techniques. This was then done by Vömel and Stüttgen (2013) for three popular memory acquisition utilities: win32dd (Suiche, 2009), WinPMEM (Cohen, 2012), and mdd (ManTech CSI Inc., 2009). Their study exhibited some correctness flaws in these tools (which were later fixed), but also showed that their level of integrity and atomicity was all quite similar.

The reason why Vömel and Stüttgen (2013) only evaluated three software-based acquisition methods lies in their measurement approach: They used the open-source Intel IA-32 emulator Bochs running a Windows XP SP3 on which the acquisition utilities ran. The utilities were instrumented such that every relevant event was recorded using a hypercall into the emulator, thus enabling the measurement. Naturally, this white-box measurement approach was only possible for tools that were available to the authors in source code, thus severely restricting the scope of their measurement. It is clear that approaches such as DMA and cold boot attacks can only be measured using a black-box approach. Furthermore, these measurements were performed in a situation where the Windows system was basically idle, thus giving a lower-bound measurement. The impact of system load on the quality of memory acquisition is not yet precisely known.

Related work

Vömel and Freiling (2012) defined correctness, atomicity and integrity as criteria for forensically-sound memory acquisition and provided a comparison matrix (Vömel and Freiling, 2011, Fig. 5) with regard to the different acquisition methods. However, they also indicate that "the exact positioning of the methods within the fields of the matrix may certainly be subject to discussion" (Vömel and Freiling, 2011, p. 7). The first to evaluate these memory acquisition criteria were Vömel and Stüttgen (2013). As already stated they relied on a white box methodology restricting them to open source tools. In 2015 Gruhn (2015) introduced a gray-box methodology with which memory address translation could be inferred. Gruhn notes the methodology can also be used to evaluate the memory snapshot correctness criteria. We build up on the results of Gruhn (2015) and extend the methodology to enable the evaluation of atomicity and integrity in addition to correctness.

Other work using the notion of atomicity are BodySnatcher (Schatz, 2007), HyperSleuth (Martignoni et al., 2010) and Vis (Yu et al., 2012), all of which try to increase atomicity of forensic memory acquisition by suspending execution of the operating system, hence reducing concurrency.

Contribution

In this paper, we present the first black-box methodology for measuring the quality of memory acquisition techniques. Extending the insights of Vömel and Stüttgen (2013), we take correctness for granted and focus on integrity and atomicity. Our approach allows to compare not only different software utilities with each other but also to compare them with totally different approaches like DMA and cold-boot attacks.

The idea of our approach is to apply the memory acquisition method to memory content that changes in a predictable way: Briefly spoken, we use a program that writes logical timestamps into memory in such a way that investigating the memory snapshot yields the precise time when a certain memory region was imaged. This allows to infer an *upper bound* in integrity and atomicity meaning that these criteria will be at most as bad for the respective procedures.

More precisely, our contributions are as follows:

- We provide a framework to evaluate memory forensic tools using a black-box approach.
- We evaluate 12 memory forensic acquisition tools and methods.

We make our tools, programs and scripts used in our evaluation available to the forensic community. This material is available at https://www1.cs.fau.de/projects/rammangler.

Outline

This paper is structured as follows: First in Section Background: criteria for forensically sound memory

snapshots we revisit the main definitions of Vömel and Freiling (2012). After this we introduce our black box measurement methodology in Section Black Box Measurement Methodology. In Section Experiments we give an overview over our experimental setup and in Section Results we outline our results. Finally in Section Conclusions and future work we conclude our work.

Background: criteria for forensically sound memory snapshots

We briefly revisit the main definitions of Vömel and Freiling (2012). In their model, memory consists of a set $\mathscr{R}$ of *memory regions* (pages, cells, or words) and a *full snapshot* covers *all* memory regions of the system, i.e., it stores a value for every memory region in $\mathscr{R}$. However, their definitions also hold for *partial snapshots*, i.e., snapshots that cover subsets $R \subset \mathscr{R}$ of all memory regions. Our evaluation methodology makes use of partial snapshots, therefore we simplify the definitions towards this case. Here we disregard correctness but rather focus on *atomicity* and *integrity*.

Atomicity of a snapshot

Intuitively, an *atomic* snapshot should not show any signs of concurrent system activity. Vömel and Freiling (2012) formalize this by reverting to the theory of distributed systems where concurrent activities are depicted using space–time diagrams. Fig. 1 shows an imaging procedure that runs in parallel to another activity on a machine using four memory regions $R = \{r_1, r_2, r_3, r_4\}$. Each horizontal line marks the evolution of each memory region over time, state changes are marked as events. The imaging procedure is shown as four events (marked as squares) that read out each memory region sequentially. Concurrently, a separate activity updates memory regions (first r_1, then r_4, then r_2, shown as black dots).

Definition 1. **(Atomicity [1]).** *A snapshot is* atomic with respect to a subset of memory regions $R \subset \mathscr{R}$ *if the corresponding cut through the […] space-time diagram is consistent.*

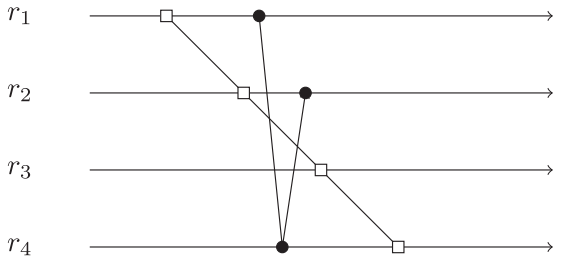


Fig. 1. Space-time diagram of an imaging procedure creating a non-atomic snapshot.

The imaging process always corresponds to a *cut* through the space-time diagram since it necessarily has to access every memory region in R. The cut distinguishes a "past" (before the snapshot) from a "future" (after the snapshot). Intuitively, a cut is consistent if there are no activities from the future that influence the past (Mattern, 1989, p. 123). Given this intuition, it is clear that the snapshot created in Fig. 1 is not atomic.

Integrity of a snapshot

Even atomic snapshots are not taken instantaneously but require a certain time period to complete. The property of *integrity* refers to this aspect. Intuitively, integrity ties a snapshot to a specific point of time chosen by the investigator. A high level of integrity implies that the snapshot was taken "very close" to that point in time.

Definition 2. **(Integrity [1]).** *Let $R \subseteq \mathscr{R}$ be a set of memory regions and t be a point in time. A snapshot s satisfies* integrity with respect to R and t *if the values of the respective memory regions that are retrieved and written out by an acquisition algorithm have not been modified after t.*

In a certain sense, integrity refers to the "stability" of a memory region's value over a certain time period. Fig. 2 illustrates the idea: The example consists again of four memory regions, i.e., $R = \{r_1, r_2, r_3, r_4\}$. We assume that at time t, the imaging operation is initiated and leads to a change in the memory regions r_3 and r_4, as indicated by the black dots (e.g., by loading a software-based imaging solution into memory). Again, the snapshot events (when the respective memory region is read out by the acquisition algorithm) are visualized as black squares. Regarding t, the snapshot satisfies integrity for memory regions r_1 and r_2 but not for r_3 and r_4.

By t, we refer to the point of time when an investigator *decides* to take an image of a computer's memory. Although being highly subjective, this point of time ideally defines the very last cohesive system state before being affected (in any way whatsoever) by the imaging operation; the value

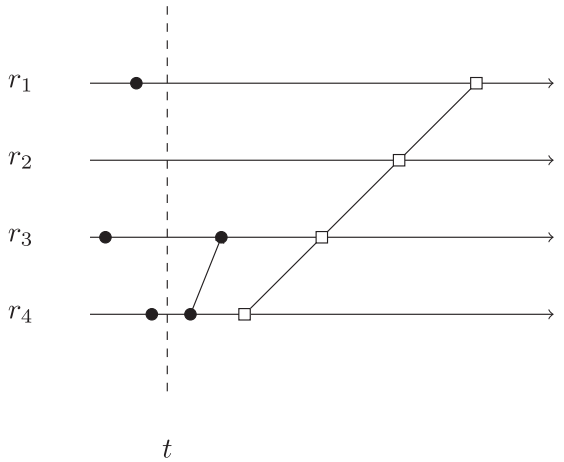


Fig. 2. Integrity of a snapshot with respect to a specific point in time t.

of t should therefore mark a time very early in the investigation process.

Vömel and Freiling (2012, Lemma 1) show that under certain assumptions the integrity of a snapshot implies its correctness and its atomicity.

Black box measurement methodology

As already suggested by Vömel and Stüttgen (2013) we developed a black box measurement methodology allowing us to estimate atomicity and integrity. Our black box methodology comprises a worst case analysis, in a high load scenario.

Implementation

Our technical implementation of the black-box methodology is as follows: First, we implement our payload application named RAMMANGL.EXE (the name refers to "RAM mangler"). This payload application allocates memory regions. Each memory region is marked with a counter which is constantly increased by the payload application like a timer.

Second, we implement an analysis framework that reads the counter value back from each region and runs statistics on them according to the estimation explained in the next section.

Estimating atomicity and integrity

We now devise two simple measures by which it is possible to estimate the atomicity and integrity of a memory snapshot. We call these measures the *atomicity delta* and *integrity delta*.

Intuitively, atomicity is bounded by possibilities to write memory regions "from the past". Hence, the faster all memory regions are acquired after the first region was "moved into the past", i.e., was acquired, the less regions can potentially be written to "from the past". Atomicity can hence be approximated by the *atomicity delta*, i.e., the interval from the acquisition of the first memory region to the acquisition of the last memory region as this is the area where inconsistencies within the image can be introduced. If no memory region has been acquired yet, all memory regions can still be freely changed because no memory regions value has been "fixed" yet. The same is true once all memory regions have been "fixed", i.e., acquired. Any changes to the memory regions then will not introduce any more inconsistencies. Good atomicity therefore corresponds to the speed of taking the memory snapshot.

In contrast, integrity implies that the memory snapshot is closely tied to the values that were present in memory at the point in time the investigator initiated the acquisition. Obviously, software-based methods must therefore have a non-zero integrity since the acquisition tool changes memory regions when loading itself into the address space of the system. Since we focus on process memory and because forensic memory acquisition tools should strive for a neglectable footprint anyway, we consider the amount of memory regions actually occupied and thereby changed by the acquisition tool to be negligible. This assumption allows us to devise the *integrity delta*, i.e., the average over the times required to acquire each memory region, given that imaging was initiated at acquisition point in time $t = 0$.

Formally, let $C = (c_0, \cdots, c_N)$ be the vector of counters embedded in the N memory regions of our payload application and t is the time the acquisition was started. We can now define our two measures that indicate atomicity and integrity as follows.

Definition 3. **(atomicity delta).** *The* atomicity delta *is the time span between the aquisition of the first memory region and the last memory region, formally*:

$$\left(\max_i c_i\right) - \left(\min_j c_j\right).$$

Definition 4. **(integrity delta).** *The* integrity delta *is the average time over all regions between starting the acquisition and acquiring that memory region, formally*:

$$\frac{\sum_{i=0}^{N} c_i}{N} - t$$

Both measures are illustrated in Fig. 3. The lower these values the better the atomicity and/or integrity respectively. Similar to the criteria of Vömel and Freiling (2012), an integrity delta of 0 implies an atomicity delta of 0.

Intuitive examples

As an intuitive example, imagine the memory of a system to consist of four memory regions, each memory region containing one counter. Initially the counters are all set to 0. Once the memory acquisition starts the counters are atomically increased every timer tick. So the ideal memory acquisition process should provide a memory image of $C = (0, 0, 0, 0)$, that is all counters of all four memory regions still being in the exact state the acquisition was started.

An example for an acquisition with high integrity but low atomicity would be the counter values $C = (0, 0, 40, 0)$. This is indicated by an high atomicity delta of 40 and a integrity delta of 10.

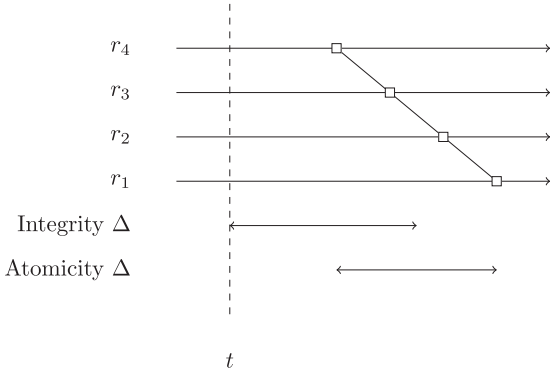


Fig. 3. Atomicity and integrity in a maximum load scenario.

An example for an acquisition with high atomicity but low integrity on the other hand would have counter values $C = \{42, 42, 42, 42\}$. In fact an atomicity delta of 0 indicates perfect atomicity.

Experiments

In this section we briefly outline our experiment. We start with the setup, then elaborate on issues we encountered during the execution. We then describe our solution to the issues and our opinion in how far this affects our results and/or forensic memory acquisition in general.

Setup

We conducted our experiments on a 64-bit Windows 7 Enterprise operating system identifiable by its build number 7600.16385.amd64fre.win7_rtm.090713-1255. The hardware used was a Fujitsu ESPRIMO P900 E90+ with an Intel i5-2400 @ 3.10 GHz and one 2 GiB RAM module. Of these 2 GiB physical RAM, 102 MiB were mapped above 4 GiB (see PCI hole discussion in Section PCI hole) and 10 MiB were lost to the BIOS and PCI devices. Resulting in a total of 2038 MiB physical RAM usable by the operating system. The pagefile of the system was disabled to ensure that all relevant data was residing in RAM.

Due to issues of Volatility relying on kernel debugging data structures, namely the kernel debugger data block (_KDDEBUGGER_DATA64), which Volatility was unable to read in all our 64-bit RAM dumps (a known issue[3]), we relied on tools made freely available[4] by Gruhn (2015) for our analysis.

The memory was acquired directly onto a removable storage device consisting of a 320 GB Western Digital WD3200AAKX-00ERMA0 hard disk inside a inateck FD2002 hard disk docking station connected via USB 2.0. Even though the docking station supports USB 3.0 it was connected via USB 2.0 only. The disk was formated with the Windows native file system NTFS. The disk connected via USB could sustain 90 MB/s write speeds. This was tested via dd. We copied (using dd) 4 GiB test files into the NTFS file system of the disk multiple times. In no memory acquisition test was this write speed reached, making us confident that disk I/O was not the determining factor of our evaluation. The memory acquisition tools were also started from this removable storage devices, which was mounted by Windows as volume E:. One exception to this was Windows Task Manager's process dumping facility, which was invoked from the system's Windows Task Manager. For the cold-boot attacks with the memimage and msramdump tools we also used an WD3200AAKX disk and the inateck USB docking station. Inception was also invoked from a lenovo X230t, the internal hard disk of which could sustain 250 MB/s write speeds again according to dd.

[3] See https://takahiroharuyama.github.io/blog/2014/01/07/64bit-big-size-ram-acquisition-problem/ and http://volatility-labs.blogspot.de/2014/01/the-secret-to-64-bit-windows-8-and-2012.html.

[4] https://www1.cs.fau.de/virma.https://www1.cs.fau.de/virma.

Sequence

We conducted our experiments as follows:

1. Startup computer.
2. Enter password to login user.
3. Wait approximately 1 min for the system to settle.
4. Open cmd.exe via the Windows startmenu.
5. Enter E: to change the working directory to our removable storage.
6. Type RAMMANGL.EXE 512 2048 but not start the payload yet.
7. Now the memory acquisition tool was started and prepared to the point where the least amount of user interaction was needed to start the acquisition. In most cases this consisted of starting another cmd.exe as Administrator, changing the current working directory to E: and then invoking the tool. In other cases we explain the procedure in subsection Analyzed methods and tools.
8. Start the payload application RAMMANGL.EXE.
9. Start the memory acquisition tool immediately afterwards. The multitude of different acquisition tools evaluated disallowed us from automating this step. We therefore tried as hard as we could to tie the start of the memory acquisition as close as possible to the point of time of starting the payload application.
10. Start timing measurement with a stop watch while not touching the system.
11. Stop time after the acquisition has completed, then close RAMMANGL.EXE (Ctrl + C).
12. Save the console output of RAMMANGL.EXE and the output of the acquisition tool to disk.
13. Note down the time on stop watch to experiment log.
14. Restart the system.

Occasionally the hard disk was defragmented to ensure maximum write capabilities. This was done at a maximum fragmentation rate of 3%, as reported by Windows. The defragmentation was also done by the Windows system.

Issues

In this subsection we will briefly list some issues that impacted our evaluation.

PCI hole

The computer system used for our evaluation remapped all memory between 2 GiB and 3.5 GiB – the so called PCI hole – to above 4 GiB. This remapping happens on the physical address level and not just the virtual address level. The remapping is usually performed by the BIOS.

This remapping was no problem for user-mode, kernel-level and physical acquisition via cold-boot attacks. It, however, posed a sometimes unresolvable problem for acquisition over DMA via IEEE 1394. IEEE 1394 only provides DMA for the lower 4 GiB of memory. In some occasions important memory structures concerning our payload process, namely the process control block, were moved to memory above 4 GiB making it impossible to

reconstruct the virtual address space of the payload process.

All of the tested kernel level acquisition tools acquire the whole address space, "including" the PCI hole remapped addresses from 2 GiB to 4 GiB, filling this non-existing RAM with zero bytes. While this is not an issue preventing our experiments, it impacts some of the measured values. Because we only consider the memory contents of our payload application and these are scattered throughout the low 2 GiB, the tools need half of their total runtime to acquire this memory anyway. For further analysis we extract all timing information from the counters in the memory region of our payload application.

Inconsistent page tables

In about every fifth memory dump acquired via kernel-level acquisition we were confronted with inconsistent page tables. While almost the whole virtual address space of our payload application RAMMANGL.EXE could be reconstructed, a few pages were sporadically mismapped to virtual memory of other processes, unused physical memory or kernel memory. The reason for this is yet unknown to us, however, because all tested kernel-level acquisition tools exhibited the same behavior, regardless of the acquisition method (either using MmMapIoSpace(), the \Device\PhysicalMemory device or PTE remapping) we do not consider it to be a tool error. However, on the other hand we also do not consider it to be an error of our framework, because we confirmed the correct assembly of our virtual address space of our payload application with Volatility and the tools provided by Gruhn (2015) [5]. To resolve this open issue we simply repeated measurements with inconsistent page tables until we acquired a correct images for our analysis.

Analyzed methods and tools

In this section we introduce the tools and methods we evaluated with our methodology.

Cold-boot attack

Cold-boot attacks were first popularized by demonstrating them practically by Halderman et al. (2009). In a cold-boot attack an attacker is leveraging the RAM's remanence effect. RAM does not lose its contents instantly but the charges of the storage capacitors rather dissiapate over time. Making it possible to read out the stored values of the RAM even after rebooting the system. To perform the attack an attacker reboots the hardware into a small operating system capable to acquire the RAM contents and write them to persistent storage (Halderman et al., 2009). Halderman et al. also made the tools, we refer to as the memimager, used for their original publication available online.[6]

Because the computer can be reset at any time, the point in time integrity is perfect. The same is true for atomicity, because once rebooted all system activity stops perfectly conserving the RAM contents. This makes the method 100% atomic. However, while the point in time of the acquisition can be chosen perfectly, the minimal operating system injected into the system inevitably overwrites memory. Because the size of the memimager is only 9.9 KiB, this RAM contents loss can in general be neglected when compared to the total 2 GiB memory size of our test system. Making the integrity also almost 100%. Another available tool to conduct cold-boot attacks is msramdump by McGrew Security.[7] It has a memory footprint of around 22.6 KiB, i.e., it will overwrite 22.6 KiB of RAM contents.

One problem with cold-boot attacks are, however, bit errors that can be introduced during hard resets of the system or when trying to transplant the RAM from one system to another (Gruhn and Müller, 2013), in which case the correctness of the acquired image is considerably impacted. Another more recent problem of cold-boot attacks is scrambling as indicated by an Intel patent (Falconer et al.) to reduce the parasitic effects of semiconductor memory. This scrambling renders cold-boot attacks involving more than a simple reset to reboot the machine futile (Gruhn and Müller, 2013; Bauer et al., 2016).

Even though the theory behind cold-booting and/or resetting a machine at a specific point already intuitively lets one assume perfect atomicity, we verified this with our experiments. Due to the introduced bit errors during a hard reset or transplantation attack, which would have a negative impact on our evaluation methodology, we refrain from such complex attacks and only performed a simple reset attack (Gruhn and Müller, 2013).

Emulation

QEMU[8] is an open source computer emulator initially developed by Fabrice Bellard. It allows operating system to be run within an emulated environment.

We called QEMU from the command line with the -monitor stdio option. This allows convenient access to the QEMU monitor on the standard terminal of the host operating system. This way the memory of the emulator can be saved by invoking the command outlined in Listing 1. During memory acquisition the emulation is paused making the acquisition fully atomic. Even though this is a known and clear cut feature of the emulator we verified the perfect correctness, atomicity and integrity of this memory acquisition method in our evaluation.

```
pmemsave 0 0x80000000 qemu.mem
```

Listing 1. Command to dump the lowest 2 GiB of memory from QEMU into the file qemu.mem.

Virtualization

VirtualBox[9] is an open source virtualization solution by Oracle. It employs Intels VT-x virtualization technology allowing a guest operating system to be run within a host operating system without the performance drawbacks incurred by pure emulation solutions. However, similar to

[5] https://www1.cs.fau.de/virma.
[6] https://citp.princeton.edu/research/memory/code/.
[7] http://mcgrewsecurity.com/oldsite/projects/msramdmp.1.html.
[8] 2.4.0 from http://www.qemu.org/.
[9] 4.3.26 from https://www.virtualbox.org/.

emulation the memory can be acquired with perfect atomicity, correctness and integrity. The command to save the entire address space of a virtual machine into an image file can be seen in Listing 2.

```
vboxmanage debugvm ${vmname} dumpguestcore --filename ram.elf64
```

Listing 2. Command to dump the physical memory from the virtual machine ${vmname} as an ELF64 dump into file ram.elf64.

Kernel-level acquisition

A very popular area of memory acquisition tools are software tools. Especially popular are so called kernel-level acquisition tools. Because in modern operating systems applications have no access to physical memory but only their own virtual address space, privileged system access from within the kernel is necessary. This is often done in form of a driver running in kernel-mode in conjunction with a user-level interface. Even though the kernel-level driver has access to the full physical memory it is hard to write memory onto disk or transmit it via the network from within the kernel. Hence, most acquisition solutions have a user-mode part used to controlling the memory acquisition driver. The user-mode part is also in charge of writing the acquired memory to an image on disk or transmitting it over the network. This can then be done with the regular APIs of the operating system.

Because kernel-level acquisition tools are essentially a part of the same system they try to acquire a forensically sound memory image from, it is a very interesting aspect how their interaction with the system impacts atomicity, integrity and correctness.

FTK Imager Lite[10] by AccessData is a graphical framework for live forensics. It supports kernel-level memory acquisition. FTK Imager was one of the closed source tools mentioned by Vömel and Stüttgen as candidate for a black-box analysis (Vömel and Stüttgen, 2013).

DumpIt[11] by Matthieu Suiche and MoonSols is another kernel-level acquisition tool. Its main feature is its simple usage. The program has no options nor configuration. The user starts it directly from a connected removable storage device. The start location is also the location to which the memory image is written. Due to this ease of use and the general availability DumpIt is rather popular.

The tool win64dd[12] is another kernel-level acquisition tool by Matthieu Suiche and MoonSols. We used the freely available Community Edition of the otherwise commercial product. Unlike DumpIt the tool win64dd offers several configuration options, such as acquisition method (MmMapIoSpace(), \Device\PhysicalMemory, and PTE remapping as default) or acquisition speed (normal, fast, sonic, and hyper sonic as default).

Winpmem[13] by Michael Cohen is an open source kernel-level memory acquisition tool. Like win64dd it offers an option to select between different acquisition modes (physical or iospace).

DMA

Memory can be acquired via a DMA attack. This attack uses a system bus such as PCI (Carrier and Grand, 2004), PCIe or IEEE 1394 (Becher et al.) to perform direct memory access (DMA) on a target machine. As stated earlier IEEE 1394 is restricted to the lower 4 GiB of physical memory and requires a software driver on the target system to be present. PCI is not hot-plugable, making it a solution that needs to be pre-installed, i.e., the target system must be made forensically ready before memory can be acquired. Because DMA has to transfer individual memory pages while the system is running, which potentially changes the memory contents, the atomicity measure, as proposed by Vömel and Freiling (2012), of this method is considered to be only moderate (Vömel and Freiling, 2011, Fig. 5).

The toolset inception[14] is a framework for DMA attacks developed by Carsten Maartmann-Moe and is available freely under the GPL license. It allows DMA attacks via IEEE 1394, also known as FireWire or i.LINK, and PCIe.

```
# ./incept dump
[...]
[\] Initializing bus and enabling SBP-2, please wait or
  press Ctrl+C
```

Listing 3. inception indicating initialization of the IEEE 1394 bus and waiting in order to enable the SBP-2.

Initializing the IEEE 1394 bus and enabling the Serial Bus Protocol 2 (SBP-2) can, according to our experiments, sometimes take up to 10 s, especially when the driver is not already loaded in the victim's machine. This is indicated by, first the output of the inception tool as can be seen in Listing 3, and second by various pop ups within the Windows operating system indicating new hardware, the initialization of new drivers and eventually the message that the new hardware is ready.

User-mode

Memory can be also acquired from the virtual memory. In such case the memory acquisition is restricted to user-level processes.

The Windows Task Manager can be used to dump memory of processes. In such case, the Windows Task Manager writes the process dump to C:\Users\user\AppData\Local\Temp\RAMMANGL.DMP. To keep the results inline with the other results the system hard disk was also a 320 GiB Western Digital WD3200AAKX hard disk. The process is suspended while the Windows Task Manager dumps the memory, resulting in high atomicity. However, because the process must be selected within the process tab of Windows Task Manager in order to initiate the dump, we exhibited a tiny lag between starting the payload application and starting the memory acquisition, slightly decreasing integrity.

[10] 3.1.1 from http://accessdata.com/product-download/digital-forensics/ftk-imager-lite-version-3.1.1.

[11] v1.3.2.20110401 from http://www.moonsols.com/2011/07/18/moonsols-dumpit-goes-mainstream/.

[12] 1.3.1.20100417 (Community Edition).

[13] 1.6.2 from https://github.com/google/rekall/releases/download/v1.3.1/winpmem_1.6.2.exe.

[14] v.0.4.0 from https://github.com/carmaa/inception.

ProcDump[15] from the Sysinternals tools is another tool that can acquire memory of a process.

```
procdump.exe -ma RAMMANGL.EXE
```

Listing 4. Commandline used to invoke ProcDump.

As can be seen from Listing 4 the process of which the memory should be acquired can be defined as the processes image name. Hence unlike previous user-mode dumping tools the lag between starting the RAMMANGL.EXE, acquiring its process ID and eventually invoking the dump tool is removed, helping greatly with the point in time integrity of dumps.

ProcDump further has various options used to trigger a dump, e.g., CPU load, memory usage or others going above a certain threshold, or the process exhibiting an exception. This gives an investigator a very high degree of fine tuning the acquisition's point in time, hence increasing integrity. ProcDump offers a method called *clone* to dump the process memory using a concept similar to copy-on-write at the operating systems level. With this it was possible to obtain a surprisingly perfect memory image, something we only expected to obtain from virtualization or emulation. Listing 5 lists the parameters used to invoke ProcDump with process cloning and reflections keeping the downtime of the process being imaged to a minimum.

```
procdump.exe -ma -a -r RAMMANGL.EXE
```

Listing 5. Invoking ProcDump to leveraging process cloning for acquisition with minimal process suspension.

Another user-level dumping tool is pmdump[16] by Arne Vidstrom. Unlike the other tools it does not suspend the process. We mostly just used it to spread the spectrum of our evaluation by introducing yet another kind of memory acquisition technique. Because the process is not suspended, pmdump exhibits memory smear resulting in reduced atomicity. Obviously, in most scenarios ProcDump leveraging cloning should be the preferred tool for acquiring a single process address space.

Results

We now present our results. We first outline our measurement accuracy. Then we give selected examples of our results. Eventually we give an overview comparison among the acquisition methods.

Measurement accuracy

We repeated all measurements until the relative standard deviation of the mean value of all counter values was below 10%. This ensures that 95% of all possible repeated measurements end up within 20% of the mean value of our measurements. Given that our objective is not to evaluate different kernel-level acquisition software with each other, as they are very close to each other as already outlined by Vömel and Stüttgen (2013), but rather to evaluate the overall state of memory acquisition these figures are good enough as each comparison group is far enough apart to not fall within 20% of one another.

Individual results

First we give a brief insight into individual results.

pmdump

Because the resulting image seen in Fig. 4 provides an expected visual, we start by introducing the results of pmdump. This also helps to explain the way in which we visualize the measurements: As can be seen from Fig. 4 it does take some time, i.e., counter increments (x-axis), until the tool starts to acquire some memory. This is first due to the fact that the tool needs the process ID of our payload application in order to dump its memory. Hence our payload application was running for some seconds during which its process ID was determined. Second, the pmdump tool seemed to be rather resource intensive slowing the overall system down, hence also slowing its own dumping process down. It can also be seen from Fig. 4 that the tool acquires the virtual address space, as the y-axis depicts the counters spread along the virtual addresses of the process.

Table 1 gives the worst case figures, i.e., the maximum figures obtained in all runs, for our atomicity and integrity delta.

Inception

Fig. 5 is quite a different picture compared to pmdump's Fig. 4. Fig. 5 shows inception in comparison to other acquisition methods. The x-axis shows time while the y-axis shows the memory regions. Each dot indicates the point in time when a specific memory region was acquired. From this it can clearly be seen that inception is the least atomic acquisition – and also the overall slowest. Because inception acquires the physical memory its acquisition plot looks rather scattered and not as orderly as the plot of pmdump. This is because Windows does not map sequential virtual addresses to sequential physical addresses. It rather keeps physical memory in a heap and allocates individual pages. Due to the memory management unit

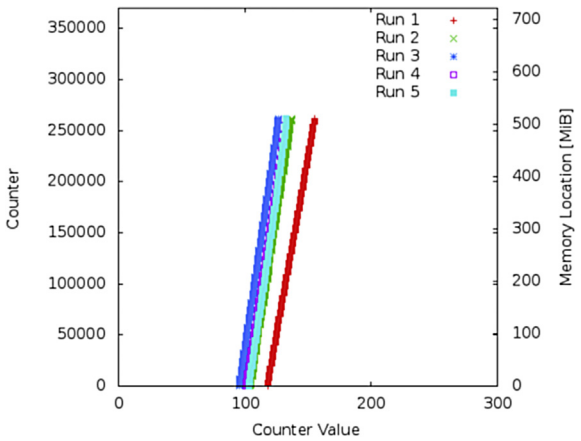


Fig. 4. Acquisition plot of pmdump.

[15] v7.1 from https://technet.microsoft.com/en-us/sysinternals.
[16] 1.2 from http://ntsecurity.nu/toolbox/pmdump/.

Table 1
Comparison of worst case atomicity and integrity deltas.

	(Worst case) Atomicity Delta	(Worst case) Integrity Delta
msramdump	1	43.84
memimager	1	63.28
VirtualBox	1	26.64
QEMU	1	35.24
ProcDump (-r)	0	39.75
ProcDump	1	36.50
Windows Task Manager	1	728.54
pmdump	37	136.62
WinPMEM	13,230	5682.24
FTK Imager	13,151	5917.24
win64dd	15,039	8077.54
win64dd (/m 1)	15,039	8172.28
DumpIt	15,711	8500.09
inception	43,898	22,056.77

(MMU) of the CPU this translation usually happens transparently without loss of performance. Again Table 1 gives the worst case figures regarding atomicity and integrity delta for inception.

DumpIt

As can be seen from Fig. 5 DumpIt is less smeared than inception via IEEE 1394. However, it is still behind super atomic methods such as cold-boot attacks and virtualization, represented by VirtualBox, which are both undistinguishable vertical lines to the far left close to the 0 point on the x-axis. We selected DumpIt here as a representative of kernel-level acquisition methods as they all are very closely related and there is no point in illustrating virtually identical acquisition methods next to each other.

Comparison

In summary it can be argued that there is quite a difference regarding atomicity and integrity considering different acquisition methods. However, as can clearly be seen from the acquisition density plot in Fig. 6 that the different methods seem to cluster. For example, the kernel-level acquisition methods are all pretty identical with regard to atomicity and integrity, as well as acquisition speeds. On other group would be DMA acquisition, here represented by the inception toolkit. Then user-mode dumping, which should be split into methods suspending process execution and methods that do not. Last but not least are the ultra high atomicity methods, which can not even be distinguished anymore in Fig. 6 because the all cluster along the y-axis. These are virtualization and emulation methods and the physical cold-boot RAM attacks.

Table 1 gives a listing of all evaluated methods worst case atomicity and integrity deltas as defined in the beginning of this paper. The table is ordered according to the above outlined groups as well as ranked by the sum of atomicity and integrity delta within each group.

Fig. 7 is the representation of Table 1 as an atomicity/integrity matrix.

Please note that the figures within Table 1 can only be compared with measurements performed with the same parameters and same hardware because the counter increments are highly hardware dependent. To compare other tools we make our framework available to the public.

Conclusions and future work

We presented a practical approach to estimating atomicity and integrity of forensic memory acquisition tools. This is the first time that this was done with a black box approach. In this way we could also test closed source software acquisition tools. We think that these evaluations are important because before conducting these experiments we had no intuition for how completely different acquisition techniques, e.g., kernel-level acquisition and DMA attacks via IEEE 1394, would compare.

Currently our results are highly tied into the hardware and RAM size. It would be worthwhile to have an independent figure representing atomicity and integrity so different tools could be compared more easily.

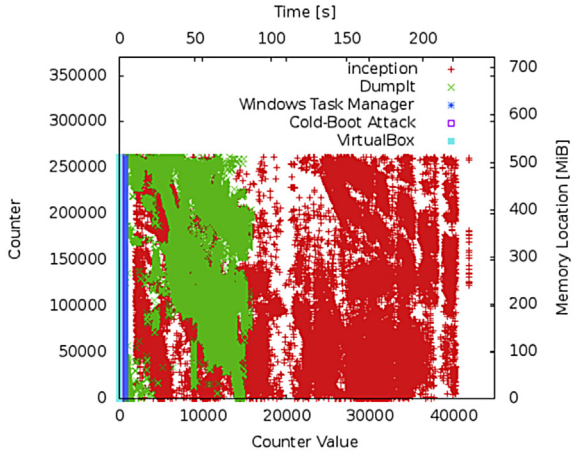


Fig. 5. Memory acquisition technique comparison (acquisition plot).

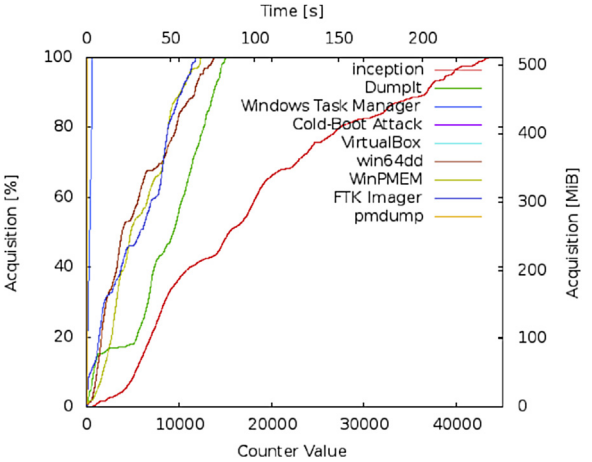


Fig. 6. Memory acquisition technique comparison (acquisition density plot).

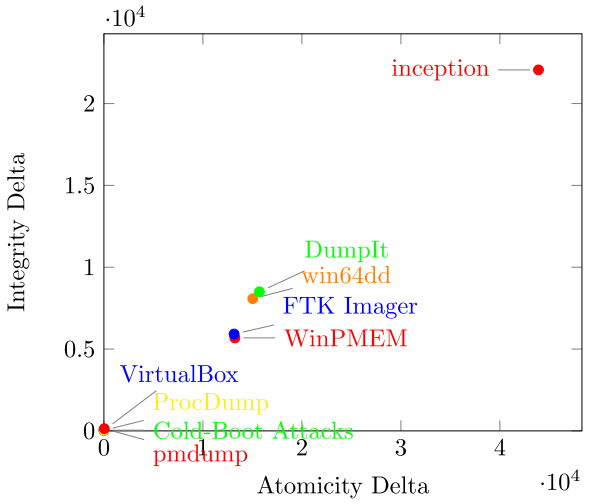


Fig. 7. Each acquisition position inside an atomicity/integrity-Matrix.

Because the impact of non-atomic memory acquisition on memory analysis is not well studied, no tools consider the issues during analysis. The impact of non-atomic, concurrent and smeared memory snapshots on forensic memory analysis are yet unknown. However, misattribution seems to be one possible issue, e.g., when a process ends while memory of the system is captured and during this time another process starts, it is possible that the memory contents belonging to these two processes get mixed up. In the worst case, incriminating evidence might be attributed to a different process and hence possibly different user. In a less worse case, exculpatory evidence may be missed, due to insufficient atomicity or integrity. Hence the impact of low atomicity and low integrity must be researched more thoroughly.

Appendix A. Supplementary data

Supplementary data related to this article can be found at http://dx.doi.org/10.1016/j.diin.2016.01.003.

References

Bauer J, Gruhn M, Freiling F. Lest we forget: cold-boot attacks on scrambled DDR3 memory. In: DFRWS EU, Lausanne, Switzerland; 2016.

Becher M, Dornseif M, Klein CN, Firewire: all your memory are belong to us. Proceedings of CanSecWest.

Carrier BD, Grand J. A hardware-based memory acquisition procedure for digital investigations. Digit Investig 2004;1(1):50–60.

Cohen M. Winpmem. 2012. URL, http://scudette.blogspot.de/2012/11/the-pmem-memory-acquisition-suite.html.

Falconer M, Mozak C, Norman A, Suppressing power supply noise using data scrambling in double data rate memory systems, US Patent 8,503,678 (Aug. 6 2013). URL https://www.google.com/patents/US8503678.

Gruhn M. Windows NT pagefile.sys virtual memory analysis. In: Ninth International Conference on IT Security Incident Management & IT Forensics, IMF 2015, Magdeburg, Germany, May 18-20, 2015; 2015. p. 3–18. http://dx.doi.org/10.1109/IMF.2015.10. URL, http://dx.doi.org/10.1109/IMF.2015.10.

Gruhn M, Müller T. On the practicability of cold boot attacks. In: Availability, Reliability and Security (ARES), 2013 Eighth International Conference on, IEEE; 2013. p. 390–7.

Halderman JA, Schoen SD, Heninger N, Clarkson W, Paul W, Calandrino JA, et al. Lest we remember: cold-boot attacks on encryption keys. Commun ACM 2009;52(5):91–8.

Libster E, Kornblum JD. A proposal for an integrated memory acquisition mechanism. SIGOPS Oper Syst Rev 2008;42(3):14–20. http://dx.doi.org/10.1145/1368506.1368510. URL, http://doi.acm.org/10.1145/1368506.1368510.

ManTech CSI, Inc.. Memory dd. 2009. URL, http://sourceforge.net/projects/mdd/files/.

Martignoni L, Fattori A, Paleari R, Cavallaro L. Live and trustworthy forensic analysis of commodity production systems. In: Recent advances in intrusion detection. Springer; 2010. p. 297–316.

Mattern F. Virtual time and global states of distributed systems. In: Workshop on parallel and distributed algorithms; 1989.

Schatz B. Bodysnatcher: towards reliable volatile memory acquisition by software. Digit Investig 2007;4(Suppl.):126–34. URL, http://dx.doi.org/10.1016/j.diin.2007.06.009. http://www.sciencedirect.com/science/article/pii/S1742287607000497.

Stüttgen J, Cohen M. Anti-forensic resilient memory acquisition. Digit Investig 2013;10:S105–15.

Suiche M. Win32dd. 2009. URL, http://www.msuiche.net/tools/win32dd-v1.2.1.20090106.zip.

Vömel S, Freiling FC. A survey of main memory acquisition and analysis techniques for the windows operating system. Digit Investig 2011;8(1):3–22.

Vömel S, Freiling FC. Correctness, atomicity, and integrity: defining criteria for forensically-sound memory acquisition. Digit Investig 2012;9(2):125–37.

Vömel S, Stüttgen J. An evaluation platform for forensic memory acquisition software. Digit Investig 2013;10:S30–40.

Yu M, Qi Z, Lin Q, Zhong X, Li B, Guan H. Vis: virtualization enhanced live forensics acquisition for native system. Digit Investig 2012;9(1):22–33.