Causal Inconsistencies Are Normal in Windows Memory Dumps (Too)

LISA RZEPKA, Universität der Bundeswehr München, Munich, Germany
JENNY OTTMANN and FELIX FREILING, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, Germany
HARALD BAIER, Universität der Bundeswehr München, Munich, Germany

Main memory contains valuable information for criminal investigations, e.g., process information or keys for disk encryption. Taking snapshots of memory is therefore common practice during a digital forensic examination. Inconsistencies in such memory dumps can, however, hamper their analysis. In this article, we perform a systematic assessment of causal inconsistencies in memory dumps taken on a Windows 10 machine using the kernel-level acquisition tool WinPmem. We use two approaches to measure the quantity of inconsistencies in Windows 10: (1) causal inconsistencies within self-injected memory data structures using a known methodology transferred from the Linux operating system, and (2) inconsistencies in the memory management data structures of the Windows kernel using a novel measurement technique based on properties of the virtual address descriptor (VAD) tree. Our evaluation is based on a dataset of more than 180 memory dumps. As a central result, both types of inconsistency measurement reveal that a high number of inconsistencies is the norm rather than the exception. We also correlate workload and execution time of the memory acquisition tool to the number of inconsistencies in the respective memory snapshot. By controlling these factors it is possible to (somewhat) control the level of inconsistencies in Windows memory dumps.

CCS Concepts: • **Applied computing** → **System forensics**;

Additional Key Words and Phrases: memory forensics, memory acquisition, system security, inconsistencies

ACM Reference format:
Lisa Rzepka, Jenny Ottmann, Felix Freiling, and Harald Baier. 2024. Causal Inconsistencies Are Normal in Windows Memory Dumps (Too). *Digit. Threat. Res. Pract.* 5, 3, Article 31 (October 2024), 20 pages.
https://doi.org/10.1145/3680293

1 Introduction

With the advent of full disk encryption [6, 15] and "file-less" malware [1], the usefulness of classical file system forensic analysis [3] is rather limited. A key issue is that critical data for forensic analysis (e.g., plaintexts, executables of malware) are not stored on persistent storage, but intentionally kept in main memory only. Therefore, the main memory of computing systems is an increasingly important source of digital evidence. This holds especially true for computers running the Windows operating system, since Windows is still one of the most popular operating systems woldwide [28].

Authors' Contact Information: Lisa Rzepka (corresponding author), Universität der Bundeswehr München, Munich, Germany; e-mail: lisa.rzepka@unibw.de; Jenny Ottmann, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, Germany; e-mail: jenny.ottmann@fau.de; Felix Freiling, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Erlangen, Germany; e-mail: felix.freiling@fau.de; Harald Baier, Universität der Bundeswehr München, Munich, Germany; e-mail: harald.baier@unibw.de.

ACM 2576-5337/2024/10-ART31
https://doi.org/10.1145/3680293

To acquire the contents of Windows main memory, multiple approaches are known, e.g., kernel-level applications, **Direct Memory Access (DMA)**, hypervisor-based approaches, or cold boot [20, 31]. While it is known that DMA- and hypervisor-based approaches lead to less inconsistencies due to their high atomicity [2, 14, 18], these techniques are not always applicable (see Section 6 for a more involved discussion of related work). Hence in practice the most popular acquisition is based on kernel-level software tools due to their widespread availability. Popular such tools are, for example, FTK Imager [10], WinPmem [29], PMDump [19], MAGNET DumpIt [12], and Memoryze [11]. However, software-based acquisition runs concurrently to normal system activity and thus with a lower atomicity level compared to hypervisor-based approaches. As a consequence they are susceptible to a phenomenon called *page smear* [4], because different parts of memory are copied at different points in time. It is therefore vital to understand and possibly even to control the amount and type of inconsistencies occurring in software-based acquired Windows memory dumps.

With respect to the Linux operating system, important results are yet available. Ottmann et al. [24] investigated the type and amount of inconsistencies that are present when performing software-based memory acquisition on Linux. As basic class of inconsistencies, they measured *causal inconsistencies* [23]. Briefly spoken, a causal inconsistency occurs if the effect of an algorithmic action is evident in the memory dump, but not its cause (a more precise definition follows in Section 2). In their experiment, they injected specific data structures with which causal relationships can be measured within the memory of a specific process. By analyzing these data structures in memory dumps, they measured the occurrence of such inconsistencies depending on various system parameters such as system load and memory size. They showed that causal inconsistencies arise frequently and that their occurrence depends on many factors, one of them being system load.

Additionally, they also used existing kernel data structures to characterize inconsistencies in memory dumps. More specifically, as suggested by Pagani et al. [25], a property of the **Virtual Memory Area (VMA)**, a kernel data structure in Linux systems to manage process address spaces, was used to characterize consistency of memory dumps. As with causal inconsistencies, the system load seems to have an impact also on the occurrence of VMA inconsistencies. However, Ottmann et al. [24] did not measure inconsistencies in Windows.

1.1 Contributions

In this article, we perform a large-scale study of inconsistencies in Windows memory dumps. Like Ottmann et al. [24], we also use self-injected causal data structures to measure causal inconsistencies, however, since memory management in Windows is substantially different from Linux, Ottmann's technique of measuring VMA inconsistencies cannot be transferred directly. We therefore need to identify similar kernel data structures in Windows to the ones used in Linux. After considering different options, we decide in favor of the Windows **Virtual Address Descriptor (VAD)** tree. The VAD is used by the Windows memory manager to store information about memory allocated by a process [9]. In our experiments, we confirm that and how the VAD tree can serve as indicator for inconsistencies in Windows memory dumps.

Overall, by using an automated approach to both memory acquisition and analysis, a large dataset of 180 memory dumps was generated and analyzed. Similar to Ottmann et al. [24], we employed a multi-threaded test program to detect causal inconsistencies and a Python script to create system load while using a memory dumping tool. Subsequently, the created memory dumps were analyzed with regard to the occurrence of inconsistencies and possible influencing factors: the execution time, the workload, and the number of threads of the test program. Our results show that both the execution time and the workload influence the number of VAD inconsistencies, while the number of threads does not seem to have an impact. Regarding causal inconsistencies, the number of active threads in the test program only has a measurable influence on the causal consistency of this part of the memory dump if at least eight threads are used. So overall, the contributions are summarized as follows:

(1) To the best of our knowledge, we are the first to use the VAD tree as an indicator to characterize inconsistencies which arise during the concurrent acquisition of main memory.

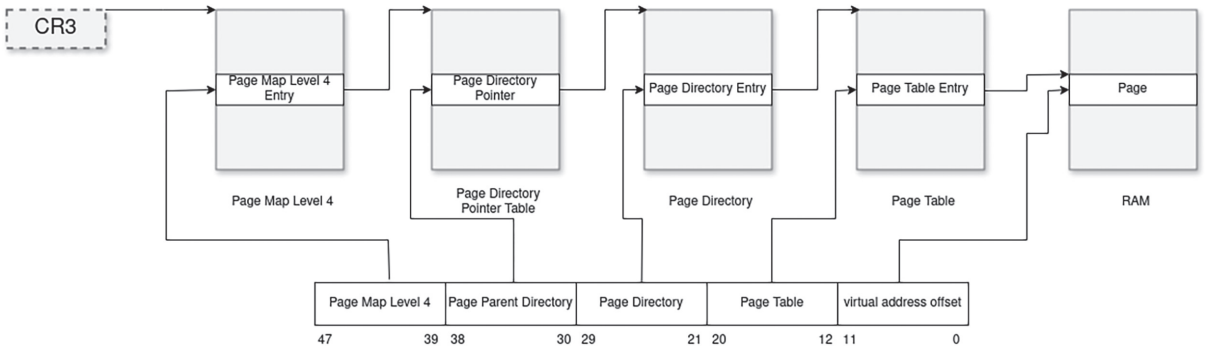


Fig. 1. Visualization of x64 virtual address translation (adapted from [33]).

(2) We developed an automation framework to acquire and analyze a large dataset of main memory dumps using WinPmem.
(3) We analyze the occurrence of inconsistencies and present the circumstances which affect the inconsistencies, i.e., number of threads, workload, and execution time.

1.2 Outline

The remainder of this paper is structured as follows: In Section 2, we present the relevant background aspects of memory management in Windows operating systems. Our evaluation approach is presented in Section 3. Afterwards, the results are presented in Section 4, where we show and explain the different experiment setups including the selected tool for memory acquisition. Finally, we discuss the results of the evaluation in Section 5. After presenting related work in Section 6, the paper is concluded in Section 7.

2 Background

This section gives an overview over memory management in Windows operating systems. Additionally, the analysis of main memory is shortly described and the notion of causal consistency is introduced. The section is concluded with an introduction to vector clocks.

2.1 Virtual Address Translation

In Windows, each process has its own virtual address space which cannot be accessed by other processes [33]. As Windows distinguishes between virtual and physical address space, it is possible to allocate more memory to a process than there actually is. In order to map virtual to physical addresses, the memory manager uses the virtual address translation. The process of address translation is dependent on the built-in processor and the architecture the processor uses, i.e., x86, x64, or ARM. In the experiment which is shown in the following section an x64 Windows 10 environment is used. For this reason, this section will explain the address translation in the x64 architecture.

A virtual address consists of 64 bits and is translated with the help of the following data structures [33]:

—Page Map Level 4 Table
—Page Parent Directory
—Page Directory
—Page Table

The translation process is shown in Figure 1. The KPROCESS data structure points to the start of the Page Map Level 4 Table. The Page Map Level 4 Table contains pointers to the Page Parent Directory. In order to find the

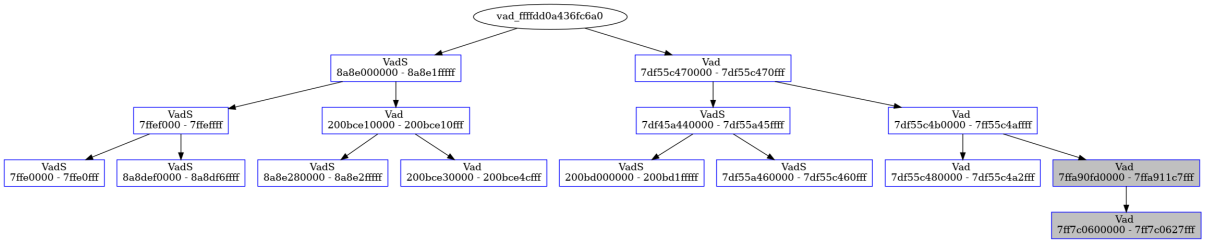


Fig. 2. Visualized VAD tree of the process smss.exe.

correct entry, the bits 47 to 39 of the virtual address are used as an index. A Page Parent Directory is similar to the Page Directory Pointer Table when using x86 translation, except the Page Parent Directory contains 512 Page Directory Pointer Tables. Next, the Page Parent Directory is searched for the needed Page Directory Pointer using the bits 38 to 30 of the virtual address. This Pointer provides the address of the Page Directory. The entries of the Page Directory are called Page Directory Entry and point to the Page Table. Here, bits 29 to 21 are used to find the corresponding Page Table Entry. The Page Table Entry, in turn, points to the start of the page in physical RAM using the bits 20 to 12. The last 12 bits of the virtual address are called *virtual address offset* [33, p. 374] and are then used in combination with the start address of the page to give the actual physical address.

2.2 Memory Management in Windows Operating Systems: The VAD Tree

The VAD tree is a data structure which is used to manage the allocated parts of the virtual address range used by a process [33, p. 401]. The tree contains multiple VAD nodes which are created by the memory manager when a process allocates memory [9]. The corresponding page directory and page table entries are only created if the process references the allocated memory which is also called demand paging [33, p. 401]. The tree is a binary self-balancing tree and consists of different node types called `MMVAD_SHORT`, `MMVAD`, and `MMVAD_LONG`. The corresponding pool tags are called VadS, Vad and VadL [9]. According to Dolan-Gavitt [9], the node types `MMVAD` and `MMVAD_LONG` are roughly the same as the `MMVAD_SHORT` node type but with additional fields. An additional field is for example a pointer to a `ControlArea` which points to `FileObjects`, i.e., mapped files or DLL files. The VAD tree is implemented using the `RTL_AVL_TABLE` structure in Windows [7]. This data structure contains a field called `NumberGenericTableElements` which gives the overall number of elements, i.e., VAD nodes, in the tree. The field is given with a variable called `VadCount` in the symbol file of the kernel.

There are different tools available to visualize the VAD tree such as Volatility or WinDbg. The work of Dolan-Gavitt produced three volatility plugins: vadwalk, vadinfo, and vaddump [9]. The plugin vadwalk traverses the VAD tree and displays it in a table as default. It is also possible to produce a Graphviz dotfile in which the tree is visualized using the plugin vadwalk or vadtree. An example can be seen in Figure 2 in which the VAD tree of the Windows process smss.exe is visualized. The tree starts with its root, the `VadRoot`. If the address range of a new node is lower than the range at the current node, the new node will be sorted left of the current node. If the address range is higher, the new node will be sorted right of the current node [9]. With the plugin vadinfo the tree is also traversed but in addition corresponding `ControlAreas` or `FileObjects` are printed. Lastly, the plugin vaddump can be used to write memory ranges to a file.

2.3 Main Memory Analysis

The analysis of acquired main memory can be divided into two approaches: *unstructured* and *structured* [24, 31]. When using the unstructured approach, the main memory copy is searched for specific strings, an approach which is problematic, since it ignores context [24]. Another downside of this approach is an accumulation of false-positives [31] as well as a high time effort. The tools using the structured approach are able to show the

context of the found information by using data structures of the operating system [24]. This allows a time efficient analysis of the acquired main memory in regard to processes, network connections, files and the system registry [31]. An example for a tool which offers functionalities for structured memory analysis is Volatility. There are two versions of the tool: *volatility*[1] which uses Python 2 (which we will refer to as *volatility2* to avoid confusion) and *volatility3*[2] which uses Python 3. In this article both versions are used, as some functionalities of plugins of volatility2 are not available in volatility3.

Volatility2 relies on profiles which need to be created by the user. The profiles contain details regarding data structures of the operating system which are used to reconstruct the to-be analyzed system [24, p. 6]. The contained information is dependent on the operating system. Throughout this work, the plugin *vaddump* of volatility2 is used. With the help of the information contained in a profile the plugin searches for the `EPROCESS` structure within the desired process address space. The `EPROCESS` structure contains the address to the `VadRoot`. From this address, the VAD tree is traversed and the plugin prints all information regarding the corresponding VAD nodes. Furthermore, the memory range given by an VAD node is extracted into a file. In contrast, volatility3 uses a program database for Windows which contains the symbol table. Additionally, volatility3 has the ability to create symbol tables based on the memory image [13].

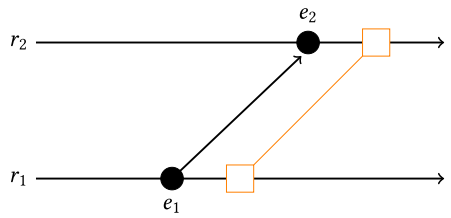
2.4 Causal Consistency

To evaluate the quality of main memory dumps, Vömel and Freiling [30] defined three formal criteria in their work: correctness, integrity, and atomicity. Correctness refers to a tool's capability to produce exact and complete copies of the memory contents. Integrity captures a different aspect, the percentage of memory contents that change between a point in time before the acquisition is started and the time at which they are copied. With this criterion, for example, the influence of the acquisition program on the memory contents can be measured. Atomicity refers to the consistency of the memory dump. Generally, a consistent memory snapshot does not contain content mismatches caused by *concurrent* activity (see Section 1). Such content mismatches, for example, cause phenomena like page smearing, where the actual contents of the physical memory pages and the state of memory as represented by the paging data structures do not match. For main memory this is of special interest, as many acquisition methods copy memory contents while other programs remain active. The evaluation of atomicity is based on cause-effect relationships between memory accesses by processes. Therefore, as proposed by Ottmann et al. [23], we use the term *causal consistency* when we refer to this criterion.

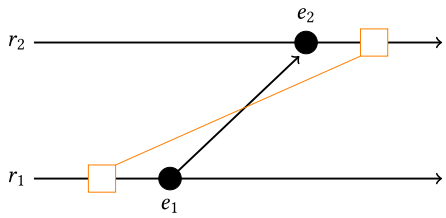
Cause-effect relationships are defined by the order in which processes perform memory accesses. An earlier access of a memory region is defined as (potential) cause of later accesses. When processes access multiple memory regions, e.g., pages, the cause-effect relationships span over the accessed regions. An example is shown in the space-time diagram in Figure 3(a). In the diagram two memory regions (r_1, r_2) are accessed by a process. The memory accesses are called events [24]. In the diagram two events occurred, e_1 and e_2. Because they were executed by the same process and event e_1 was executed before e_2, e_1 is the cause of e_2. In the diagram, the arrow from e_1 to e_2 indicates this. The memory acquisition process is depicted using orange rectangles. As both events are contained in the memory snapshot, the snapshot is causally consistent regarding events e_1 and e_2.

In contrast, if e_1 was missing from the snapshot, it would be causally inconsistent. In this case, depicted in Figure 3(b), information needed for the interpretation of the memory contents is absent. As an example let us assume that the first event sets a pointer saved in memory region r_1 to the address of r_2 and the second event changes the contents of the latter region. If the result of the first event (the reference to r_2) is missing in the snapshot, the content of r_2 cannot be connected to the right pointer and might be overlooked during the analysis.

[1] https://github.com/volatilityfoundation/volatility
[2] https://github.com/volatilityfoundation/volatility3



(a) Space-time diagram of a causally consistent memory snapshot. (adjusted from [23])



(b) Space-time diagram of a causally inconsistent memory snapshot. (adjusted from [23])

Fig. 3. Space-Time-Diagrams of causally consistent and inconsistent memory dumps.

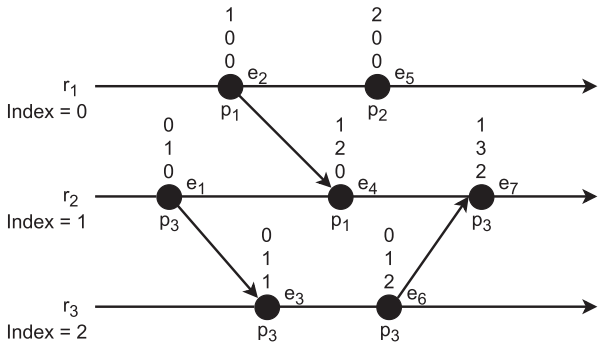


Fig. 4. Vector clocks are used to track cause-effect relationships. Each time an event is executed, the local event counter for the memory region is increased. When a process accesses a region its saves the updated vector clock. The saved vector clock is merged with the current vector clock of the next region it accesses, for each index in the vector the highest value is kept. (adjusted from [23])

2.5 Vector Clocks

Observing causal inconsistencies requires tracking the cause-effect relationships between memory regions. In our evaluation we use the same observation method as Ottmann et al. [23] which relies on *vector clocks* [22], a concept from distributed computing, for this purpose.

The first step for constructing vector clocks is an event counter on each memory region. This counter is increased by one each time an event, i.e., a write access to the region by a process, occurs. Thereby, an order of events on one region can be established with events with lower counters having occurred before events with higher counters. This can be seen in Figure 4 on region r_1. With the first event on the region, e_2, its counter is set to 1, the second event, e_5 increases it to 2. To track cause-effect relationships between memory accesses such an order of events needs to be established over all memory regions. Therefore, for each region not only the local event counter is saved but a vector with an index for each memory region in the observed memory range. In the example shown in Figure 4, the vectors have three indices since three memory regions are observed. The local counter for each region in its vector is located at the region's index.

To establish the event order, whenever a process accesses a memory region it saves the region's current vector clock. When it performs its next access the saved vector clock and the vector clock of the currently accessed region are compared and for each memory region the highest counter is kept. For example, in Figure 4, event e_6 is executed by process p_3 on region r_3 and event e_7 is executed by the same process on region r_2. After the first event, the process saves the vector (012). Therefore, when the second event is executed the previous vector of

region r_2, (120), and this vector are combined. This results in the vector (122). Lastly, the region's local counter is increased by one, resulting in (132) as the region's vector clock's new value.

As with the local counters, when two vector clocks are compared, the one with the smaller values represents events executed before the one with the higher values. This is true, for example, for events e_6 and e_7. After both events have been executed, for all indices the values saved in r_3's vector clock are smaller than the ones saved in r_2's vector clock. Therefore, the vector clocks allow us to infer that e_6 was executed before e_7. When two memory regions are not connected by a cause-effect relationship this also becomes apparent through comparison. For example, event e_5 does not have any connection to another memory region, i.e., in Figure 4, no arrow from another memory region to region r_1 before the event was executed is present. Therefore, when we compare r_1's vector after e_5 with the other region's vector clocks at any given time no clear order can be established since it always has the higher value for index 0 but a lower value for the other indices. Whenever this is the case, the memory access is independent of the other observed ones.

3 Methodology

This section describes the methodology used in the evaluation. This includes the consistency indicators used throughout this work as well as the evaluation procedure.

3.1 Consistency Indicators

In order to assess the consistency of acquired main memory dumps, indicators are needed. Similar to the evaluation by Ottmann et al. [24] on Linux, we use two approaches to track inconsistencies. First, we measure causal inconsistencies in the heap of the test program, called pivot program. Second, we observe mismatches in an operating system management structure, the VAD tree.

3.1.1 Pivot Program. In order to visualize causal inconsistencies within self-injected memory data structures, a software called *pivot program* is used which was previously introduced by Ottmann et al. [24]. The pivot program incorporates the notion of vector clocks, see Section 2.5, to track causal inconsistencies in the pivot program's heap. In a list, each element possesses a vector clock which is updated when a thread of the program removes or inserts an element [24]. A vector contains a row for every memory region. The value at the index of the row, which is assigned to a memory region, represents the *local counter* and is incremented with every event that is happening in the said memory region [24, p. 8]. In order to get information about the local counters of other memory regions, every process stores information about the vector of the memory region in which the process made an action. The local counter of the memory region is then compared with the vector stored in the process and later the local counter is updated to keep track of the other memory region's counter. Lastly, the *global time vector* is calculated by writing the highest value for each index in each memory region into the vector. Afterwards, the values at all indices are compared with the local counter of the corresponding memory region. If these values differ, an inconsistency is found. As the program was originally written for a Linux operating system, cygwin[3] is used to compile the program for the evaluation on Windows.

3.1.2 VAD. As explained in Section 2.2, the VAD tree is implemented as a structure called RTL_AVL_TABLE and holds a field which gives the overall number of VAD nodes in the tree, called `VadCount` in the symbol table of the kernel. This variable is compared with manually counting all the nodes in the tree. If the two numbers are not equal, there is an inconsistency in the memory dump. This does not allow conclusions about possible consequences for the analysis of the acquired memory dump. The described method is implemented using a plugin for volatility3. Here, the plugin vadinfo is adapted to print the difference in the number of VAD nodes in a process. The plugin compares the `VadCount` variable with a counter which counts all nodes in the VAD tree while

[3] https://www.cygwin.com/

Table 1. The Definition of the Three Activity Levels for the Evaluation

Activity level	Memory usage (%)
0	25
1	35–40
2	50–60

the plugin traverses it in order to print information about the nodes. The plugin prints all found inconsistencies, i.e., a difference between the variable and the count of nodes in the tree.

In order to verify whether the `VadCount` variable and the count of nodes includes or excludes the root of the VAD tree, an additional experiment is conducted. In this experiment, a browser is open while the memory is acquired, as a browser potentially creates inconsistencies. Then, the virtual machine is paused to acquire the main memory. This is done five times. Afterwards, volatility3 and the written plugin for comparison of VADs is used to analyze the memory dumps. There are no VAD inconsistencies in all five acquired memory dumps of the paused virtual machine which indicates that the variable `VadCount` and the count of the nodes treat the `VadRoot` in the same way. Additionally, the results showed that the comparison works in the intended way.

3.2 Evaluation Process

In the evaluation a virtual machine with Windows 10, Version 22H2 is used. The virtual machine employs four CPUs, 8 GB of RAM and runs on a physical computer with an Intel Core i7-2600 processor and 16 GB RAM. There are no other virtual machines running. The virtual machine is set up using VirtualBox. The evaluation consists of three activity levels and for each level 60 memory dumps are generated. In particular, the mentioned pivot program is started with one, two, and four threads in order to analyze whether the number of causal inconsistencies is dependent on the number of threads. For each combination of activity level and number of threads 20 memory dumps are created. The activity levels differ in the overall memory usage (Table 1). For the memory acquisition process with activity level 1 and 2, additional workload is generated using a python script which starts different Windows applications, e.g., the Firefox browser with a number of open tabs and the Microsoft Store. During the acquisition the memory usage is checked using a PowerShell script. By the use of the script differences in the memory usage for the same level could be identified, although the same applications were called in each iteration of the scenario. For this reason the memory usage of level 1 and level 2 is given as an interval in Table 1.

The tool WinPmem is used to acquire the main memory. In a short preliminary study we tested different available tools (FTK Imager, Magnet RAM Capture, Belkasoft RAM Capturer, and WinPmem) regarding availability, malfunction, and execution time. As WinPmem is open source as well as command line based and has in comparison a short execution time, we selected WinPmem for the evaluation process. For each memory dump, a scenario similar to [24] is used which is depicted in Figure 5: The memory acquisition process is automated using Python and PowerShell. The Python script starts the pivot program with the according number of threads as well as additional workload. Within each generated memory dump, the number of VADs is compared to find inconsistencies using the previously mentioned volatility3 plugin. Afterwards, the heap of the pivot program is extracted from the memory dump using the volatility2 plugin vaddump which is adapted to dump only the memory range in which the list elements of the pivot program are resident. Then, a python script written in the preceding work of Ottmann et al. [24] is used to find causal inconsistencies in the pivot program's heap.

4 Evaluation

We now discuss the results of our experiments. As the execution time potentially influences the number of inconsistencies, we first show results regarding the influence of the activity levels and number of threads on the

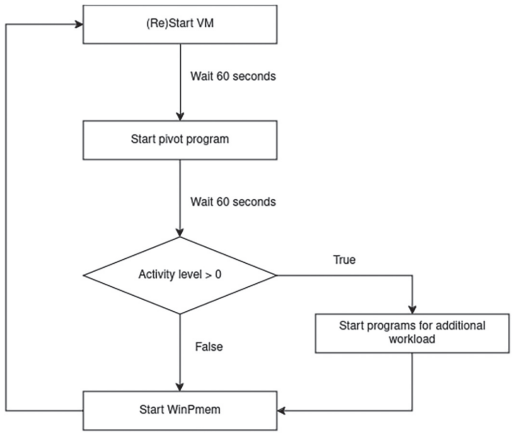


Fig. 5. Scenario for each iteration of the evaluation.

Table 2. Mean Execution Time and Standard Deviation (σ) of WinPmem in Minutes Per Activity Level and Number of Threads of the Pivot Program

	1 Thread		2 Threads		4 Threads	
	Mean execution time (min)	σ	Mean execution time (min)	σ	Mean execution time	σ
Level 0	1.68	0.23	2.11	0.24	2.24	0.28
Level 1	2.34	0.26	2.50	0.37	2.62	0.23
Level 2	2.93	0.29	3.28	0.45	3.33	0.82

execution time of WinPmem. Afterwards, the occurrence of VAD inconsistencies and their influencing factors are analyzed. Lastly, the occurrence of causal inconsistencies is inspected.

4.1 Execution Time

The execution times of WinPmem for different parameters are shown in Table 2 and depicted in Figure 6. The execution time of WinPmem is captured using the PowerShell command *Measure-Command*. The results show that the execution time of WinPmem increases with each activity level, i.e., with a higher memory usage. This is also visible when looking at the median: for activity level 0 the median is 2.01 minutes, 2.5 minutes for activity level 1 and for activity level 2 the median is 3.14 minutes. There is no outlier in activity level 0. The one outlier of activity level 1 shows a much shorter execution time. Activity level 2 exhibits three outliers of which two show a shorter and one shows a longer execution time than the median. The increasing execution time can also be observed in Table 2. For each activity level, the mean execution time of WinPmem increases. Moreover, Table 2 shows the standard deviation for each mean of each activity level. The execution time of WinPmem not only increases per activity level but also per number of threads of the pivot program. This can be seen in Figure 6. Table 2 supports this observation.

4.2 VAD Inconsistencies

The number of VAD inconsistencies increases with the activity level and the number of threads of the pivot program. This is shown in Table 3. According to the table, there is a memory dump which is listed as not containing any VAD inconsistencies and was acquired with activity level 0 and 4 threads.

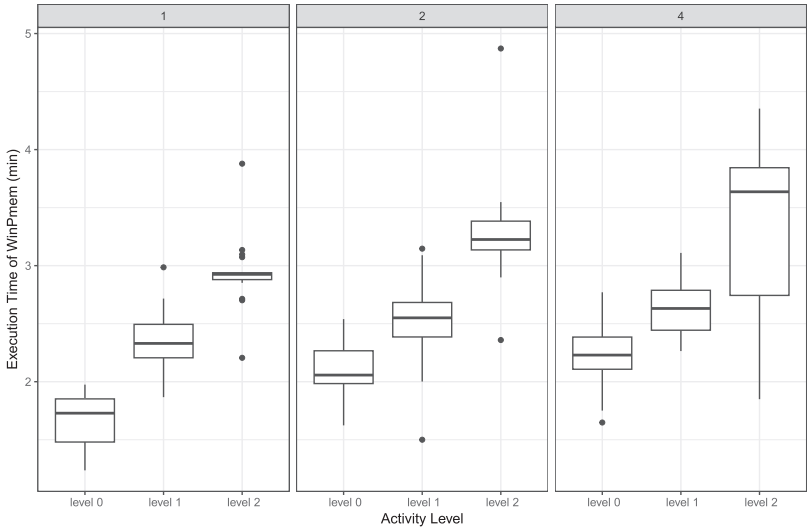


Fig. 6. The execution time of WinPmem grouped by the different activity levels and separated by the number of threads of the pivot program.

Table 3. Number of VAD Inconsistencies and Number of Memory Dumps Which Contained the VAD Inconsistencies Per Activity Level and Number of Threads

	1 Thread		2 Threads		4 Threads	
	VAD Incon	Mem dumps	VAD Incon	Mem dumps	VAD Incon	Mem dumps
Level 0	1,773	20/20	2,386	20/20	3,141	19/**19**
Level 1	5,372	20/20	6,762	20/20	7,919	20/20
Level 2	15,308	20/20	18,090	20/20	24,371	20/20

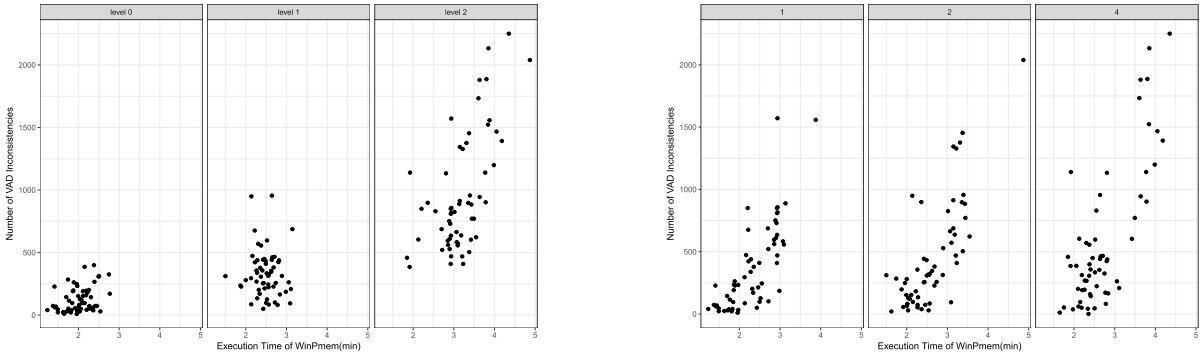
The relationship between the occurrence of VAD inconsistencies and the execution time separated by activity level or threads is shown in Figure 7(a) and (b). Additionally, the spread of the VAD inconsistencies can be observed. Only the activity level seems to have an impact on the number of VAD inconsistencies.

In order to analyze whether the execution time and the number of VAD inconsistencies are related, a correlation test is conducted. For further information regarding the statistical tests used throughout this work, please refer to Appendix A. The p-value which is smaller than 0.001 indicates that the relationship between the execution time and the number of inconsistencies is significant, i.e., the relationship exists not by chance. The R value which is 0.72 describes that there is a positive relation as well as a large effect size [5].

Additionally, the Spearman correlation between the different activity levels and the number of VAD inconsistencies is calculated. The relationship seems to be significant which is supported by the p-value. There is also a positive relation, as the R-value is 0.85.

4.3 Causal Inconsistencies

The number of causal inconsistencies increases with the activity level. This is shown in Table 4. As the heap could not be extracted for three memory dumps in activity level 0 with 2 threads, activity level 1 with 2 threads, and activity level 2 with 8 threads, they are excluded from further analysis. The mentioned memory dumps are marked



(a) WinPmem execution time (min) and the number of VAD inconsistencies separated by activity levels.

(b) WinPmem execution time (min) and the number of VAD inconsistencies separated by number of threads.

Fig. 7. Relationship of the WinPmem execution time and the number of VAD inconsistencies.

Table 4. Number of Causal Inconsistencies and Number of Memory Dumps Which Contained the Causal Inconsistencies Per Activity Level and Number of Threads

	1 Thread		2 Threads		4 Threads		8 Threads	
	Causal Incon	Mem dumps	Causal Incon	Mem dumps	Causal Incon	Mem dumps	Causal Incon	Mem dumps
Level 0	104	20/20	133	18/**19**	104	15/20	435	20/20
Level 1	85	11/20	192	12/**19**	114	9/20	440	20/20
Level 2	43	10/20	201	16/20	298	18/20	886	19/**19**

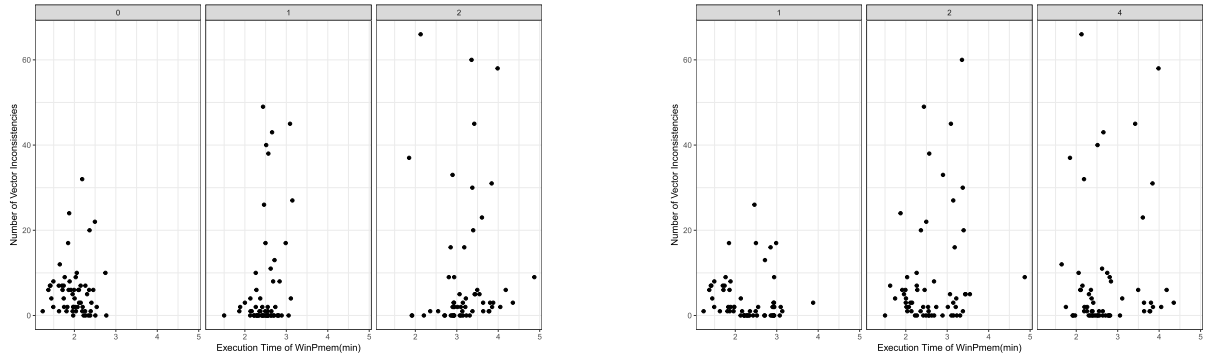
in Table 4. The number of threads does not seem to have an impact on the occurrence of causal inconsistencies when the pivot program is started with 1, 2, and 4 threads. The number of inconsistencies is the highest for each activity level when the pivot program is started with 8 threads.

Figure 8(a) and (b) show the relationship between the occurrence of causal inconsistencies and the execution time of WinPmem in minutes separated by activity level or threads. Additionally, the spread of the causal inconsistencies can be observed. Both activity level and threads do not seem to have an impact on the number of causal inconsistencies.

In order to analyze the relationship between the WinPmem execution time and the number of causal inconsistencies, a Spearman correlation test needs to be conducted. According to the p-value, which is greater than 0.05, the relationship is not significant. This observation implies that the relationship between the execution time and the number of causal inconsistencies may be by chance. The R-value which is close to zero (−0.01) indicates a very weak relationship between the two variables.

4.3.1 Results with More Threads. As the results showed that the number of threads surprisingly does not have an impact on the occurrence of causal inconsistencies (see Section 5.3), an additional experiment is conducted in which the number of threads of the pivot program is increased to eight. For this experiment, 60 additional memory dumps are acquired, i.e., 20 dumps per activity level. Afterwards, the analysis is conducted with this additional group to show whether the number of threads was too small to show a difference.

Table 4 shows the overall number of inconsistencies per activity level and for the pivot program started with the according number of threads, including the additional experiment with eight threads. With eight threads, there are inconsistencies in all memory dumps except for one. The dump which did not contain any inconsistencies was further investigated with the plugin pslist of volatility3. The plugin showed that the pivot process was not



(a) WinPmem execution time (min) and the number of causal inconsistencies separated by activity levels.

(b) WinPmem execution time (min) and the number of causal inconsistencies separated by number of threads.

Fig. 8. Relationship of the WinPmem execution time and the number of causal inconsistencies.

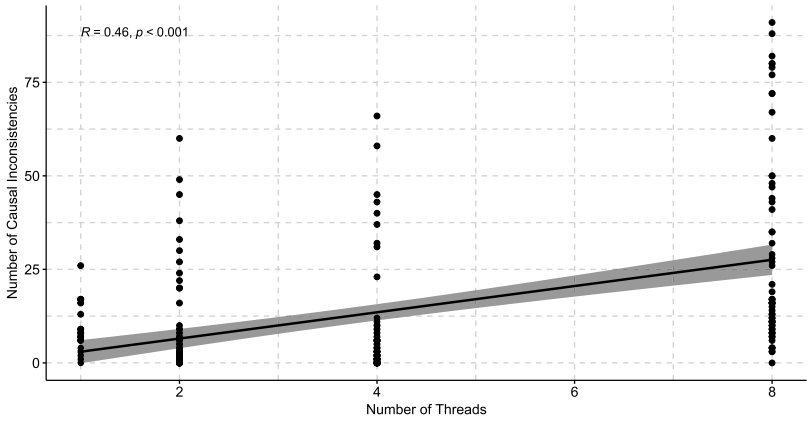


Fig. 9. Spearman correlation between the number of causal inconsistencies and the different number of threads including the additional experiment.

in the process list. A possible explanation is a termination of the pivot program before the memory acquisition process is complete.

Another observation is that the number of causal inconsistencies is much higher when the pivot program was started with eight threads than with the other number of threads. This behavior can also be observed in Figure 9. The Figure shows the spread of the causal inconsistencies per number of threads. There seems to be more inconsistencies with eight threads. For this reason, the hypothesis that the number of threads was too small to show a relationship seems to hold. In order to investigate this observation further, a correlation test needs to be conducted. Therefore, a Spearman correlation is conducted [27]. The correlation between the number of threads and the number of causal inconsistencies has a medium effect size, as the R-value is 0.46 [5]. Additionally, the relationship is statistically significant (p-value < 0.001).

A Kruskal–Wallis rank sum test shows differences between the four groups (p-value $= 6.47 \times 10^{-16}$). Therefore, a Wilcoxon rank sum test is conducted as a *post hoc* test [32]. As the results now indicate a relationship between the two variables, there should also be differences between the groups compared to the previous experiment. The test shows that there are only differences between each of the groups with one, two, and four threads compared

to the group with eight threads. This verifies the previous observation which showed no differences between one, two, and four threads.

5 Discussion

This section discusses the experimental results presented in Section 4.

5.1 Execution Time

Due to the results of the evaluation, it is assumed that both activity level and execution time have an impact on the occurrence of VAD inconsistencies. For this reason, the relationship between execution time and activity level is investigated in this section. Additionally, the relationship between the number of threads of the pivot program and the execution time is analyzed.

5.1.1 Activity Levels. According to the results in the previous section, the execution time increases with the activity level. The mentioned outliers in the box plot are most likely due to the difference in background applications which are started in each iteration of the dumping process, which was already mentioned in the previous section. Table 6 supports the increasing execution time observation, as the mean execution time also increases for each activity level.

In order to analyze the relationship between the two variables, a Spearman correlation test is conducted. The p-value of the correlation test is smaller than 0.05 which means that the relationship between the execution time and the activity levels is significant, i.e., not by chance. The R-value which is 0.77 indicates a positive relationship. According to Cohen [5], everything greater than 0.5 indicates a strong correlation.

As the residuals are neither following a normal distribution nor the variance is homogeneous, a Kruskal–Wallis test is conducted to determine if there are differences between the activity levels while using all numbers of threads [21]. The test yields a p-value smaller than 0.05. This indicates that there are differences between the activity levels, i.e., the groups are not from the same population. The test does not provide information about between which activity levels the differences are. For this reason, a pairwise Wilcoxon rank sum test is used as a *post hoc* test [32]. The test is conducted using the Bonferroni correction to minimize the alpha error, as the error is more probable when testing the same sample multiple times [32]. As the p-value is smaller than 0.05 for each comparison, all activity levels are significantly distinct from each other regarding the execution time.

5.1.2 Threads. As the results of the evaluation show an increase of the execution time per number of threads, it is also verified whether a relationship between the two variables exists.

As the residuals are not following a normal distribution, a Spearman correlation test is conducted to verify a relationship between the execution time and the number of threads [27]. The p-value of this test is smaller than 0.05 which means that it is statistically significant, i.e., there is a correlation between the execution time and the number of threads. In contrast, the R-value is 0.22 which is only a weak effect according to Cohen [5]. In particular, when comparing the R-value of the results with the number of threads with the R-value of the results with the different activity levels, the correlation between the activity levels and the execution time is stronger than between the execution time and the number of threads.

A subsequent Kruskal–Wallis test yields a p-value of 0.006173 which is smaller than 0.05. Therefore, there are differences between the three groups and a pairwise Wilcoxon rank sum test is conducted as a *post hoc* test. The p-values of the comparison between one and two and additionally between one and four threads are smaller than 0.05 and thus statistically significant, i.e., there are differences between the groups. Between two and four threads, the p-value is 1 which means that there is no difference between them.

As a summary, the activity level seems to have a stronger impact on the execution time than the number of threads of the pivot program. This is supported by the different R-values which indicates a stronger correlation between the activity level and the execution time than between the number of threads and the execution time.

PID	Process	VadCount	Count	Difference
476	csrss.exe	138	140	2
552	csrss.exe	99	90	9
844	svchost.exe	160	148	12
964	svchost.exe	99	92	7
1264	svchost.exe	81	80	1
3684	svchost.exe	77	75	2
4176	StartMenuExper	214	213	1
5668	RuntimeBroker.	117	119	2
7072	svchost.exe	195	194	1
2008	PhoneExperienc	663	652	11
1312	powershell.exe	181	183	2
5984	RuntimeBroker.	112	109	3
15049565405232	I	1740641255981842483	0	1740641255981842483

Fig. 10. Corrupted process in the output of the plugin vaddiff.py.

Table 5. Mean of All Processes Listed in the Output of the Volatility3 Plugin Pslist, Mean of Processes Which Contained VAD Inconsistencies and Mean of the Number of VAD Inconsistencies of 60 Memory Dumps Per Activity Level

	Mean of all processes	Mean of processes with VAD inconsistencies	Mean of VAD inconsistencies
Level 0	113	17	122
Level 1	151	51	334
Level 2	172	80	963

Additionally, the *post hoc* Wilcoxon rank sum test showed that there are significant differences between all three activity levels. In contrast, the test yields differences only between two groups when using the number of threads as the group variable. This behavior was assumed, as with an increase of the activity level more applications are started which run concurrently while acquiring the main memory. For this reason, WinPmem has to share more CPU time which leads to a longer time to acquire the main memory.

5.2 VAD Inconsistencies

As mentioned in Table 3 of the previous section, one memory dump was excluded from the analysis regarding VAD inconsistencies. Figure 10 shows a part of the processes which are listed when using the VAD comparison plugin of volatility. The process which is listed at the end of the output stands out, as the columns are filled with numbers rather than for example the name of the process or the PID. Additionally, the columns which represent the number of VAD nodes and the VAD count yield large numbers. As this would falsify the overall number of VAD inconsistencies, this memory dump is excluded when adding up all VAD inconsistencies. The same applies to Table 3.

5.2.1 Execution Time. The results show that there seems to be a strong correlation between the execution time and the number of VAD inconsistencies which is supported by the R-value of 0.72 [5]. With a longer execution time and concurrently working processes, the number of VAD inconsistencies increases. Table 5 shows that Windows has many background processes. The VAD inconsistencies mostly derive of these processes. Additionally, the mean of processes which contain VAD inconsistencies increases with each activity level. In particular, when the activity level is increased from 1 to 2, the number of processes with VAD inconsistencies is larger than the number of added processes.

5.2.2 Activity Levels. Table 3 shows that with an increase of the activity level, the VAD inconsistencies are more than doubled. A reason for this behavior could be the additional applications which are started with an increasing activity level. With more processes running concurrently while acquiring the main memory, it is more likely that a process' pages are changed in between the acquisition process which leads to inconsistencies. The R-value of 0.85 indicates a strong correlation [5].

The p-value smaller than 0.05 of a Kruskal–Wallis rank sum test indicates that there are differences between the activity levels regarding the number of VAD inconsistencies. As the p-value of the subsequent pairwise Wilcoxon rank sum test is smaller than 0.05 in all combinations, there are significant differences between all three activity levels regarding the number of VAD inconsistencies.

5.2.3 Threads. According to Figure 7(b), the number of threads of the pivot program does not seem to have an impact on the overall number of VAD inconsistencies. As the residuals are not following a normal distribution, the Spearman correlation method is used to verify this. The p-value yields 0.024. Hence, there is a relationship between the number of VAD inconsistencies and the number of threads of the pivot program. However, the R-value which is 0.17 indicates that there is only a very weak correlation between the two variables.

A following Kruskal–Wallis test yields a p-value of 0.1157 which indicates that there are no differences between the groups, i.e., the different number of threads, and the number of VAD inconsistencies [32]. In particular, it does not matter whether the pivot program is started with one, two or four threads regarding the number of VAD inconsistencies, as the means of the inconsistencies for each group do not differ. In conclusion, this supports the observation that the number of threads does not seem to have an impact on the overall number of VAD inconsistencies.

As a summary, both execution time and activity level have an impact on the number of VAD inconsistencies. With a longer execution time it is more likely that the pages of many background processes of Windows change while acquiring the main memory. Additionally, a higher number of applications which run concurrently increases the occurrence of VAD inconsistencies. In contrast, the number of threads of the pivot program does not seem to have an impact on the overall number of VAD inconsistencies. The Spearman correlation yields a very weak effect. A reason could be many outliers which can be seen in Figure 7(b). Additionally, the means of the inconsistencies per thread number do not differ. In order to research the weak relationship between the number of threads of the pivot program and the number of VAD inconsistencies, only the VAD inconsistencies of the pivot program must be included when conducting an additional correlation test.

5.3 Causal Inconsistencies

Table 4 of the previous chapter shows that with activity levels 0 and 1 19 instead of 20 memory dumps are used as the total number of memory dumps. When looking at the two memory dumps with the plugin pslist of volatility, the pivot program is not shown in the list of processes. For this reason, these memory dumps are excluded from the analysis.

5.3.1 Execution Time. The results of the correlation test between the execution time of WinPmem and the number of causal inconsistencies indicate that there is no relationship between the execution time of WinPmem and the number of causal inconsistencies. This is surprising because with a longer execution time more pages can be changed on the heap which leads to a higher probability to encounter inconsistencies.

5.3.2 Activity Levels. In Figures 8(a) and (b) the spread of the causal inconsistencies grouped by the activity level and threads is shown. Both scatter plots indicate that there is no relationship between the number of causal inconsistencies and the activity level or number of threads. In order to analyze whether this assumption holds, a Spearman correlation test is conducted.

As the p-value is 0.17 and therefore greater than 0.05, there is no correlation between the number of causal inconsistencies and the activity level. This result supports the assumption that with higher kernel activity the

heap of the pivot process is not more frequently changed, and therefore the activity level should not have an impact on the number of causal inconsistencies [24].

To verify whether there are differences between the activity levels regarding the number of causal inconsistencies, a Kruskal–Wallis rank sum test is conducted [32]. As the p-value is 0.006193, there are differences between the groups. Therefore, a pairwise Wilcoxon rank sum test is conducted as a *post hoc* test [32]. Only the p-value between activity level 0 and 1 is smaller than 0.05. For this reason, there is only a difference between activity level 0 and activity level 1 regarding the number of causal inconsistencies.

5.3.3 Threads. When the pivot program is started with additional threads, the heap of the pivot process can be changed more frequently due to concurrent activity [24]. For this reason, it is assumed that the number of threads have an impact on the number of causal inconsistencies. The scatter plot in Figure 8(b) does not support this assumption, as the number of causal inconsistencies does not seem to increase per thread number. This can also be observed in Table 4 for 1, 2, and 4 threads. For an increasing thread number, the number of causal inconsistencies does not necessarily also increase. This holds true for activity level 0 and activity level 1. Only activity level 2 supports the assumption that with an increasing number of threads, the number of inconsistencies also increases.

In order to verify whether the thread number has an impact on the number of inconsistencies, a Spearman correlation test needs to be conducted. As the p-value of the correlation test is 0.342, there seems to be no relationship between the number of threads and the number of causal inconsistencies when using 1, 2, or 4 threads for the pivot program.

In contrast, the correlation test of the additional experiment with the pivot program started with eight threads shows that there is a relationship between the number of threads and the number of causal inconsistencies (Section 4.3.1). This contradicts the previous results and supports the assumption that the number of threads was too small to show differences between the groups. With a higher number of threads, the heap of the pivot program is more frequently changed which leads to a higher occurrence of causal inconsistencies.

In conclusion, the execution time of WinPmem does not seem to have an impact on the number of causal inconsistencies. This is surprising because with a longer execution time, the pivot process' heap is more frequently changed and therefore inconsistencies are more likely to happen. The relationship between the number of causal inconsistencies and the number of threads of the pivot program is only visible when using at least eight threads. Furthermore, the activity level does not seem to have an impact on the number of causal inconsistencies. The results support the assumption that the pivot program's heap is not affected by higher kernel activity.

6 Related Work

Memory can be acquired from different levels within the system hierarchy [20]. Depending on the level it might be possible to halt concurrent activity on the system during the acquisition. For example, when acquiring the memory contents of a virtual machine using its hypervisor's capabilities, the execution of the virtual machine can be paused while the memory is copied. However, if this is not possible, concurrent activity can lead to content mismatches within the copy. This is what we refer to as inconsistency. Moreover, an acquisition tool that is executed within the system of which memory is acquired likely influences the memory contents. For example, acquisition tools at the level of the operating system use main memory for their execution. Therefore, the quality of a memory acquisition tool depends on multiple factors. Several approaches exist for capturing these factors in evaluation criteria [2, 16, 26, 30].

Our work is mainly related to evaluations of the quality of memory acquisition tools operating at the kernel level used on Windows and, as we use the same evaluation procedure, the evaluation presented by Ottmann et al. [24] for a memory acquisition tool used on Linux.

6.1 Evaluations on Windows

Campbell [2] analyzed the impact of four memory acquisition tools on the memory contents and their accuracy. The four tested tools are Windows Memory Reader, WinPmem, FTK Imager, and DumpIt. In the experiment, two virtual machines with Windows 7 as an operating system are created. One virtual machine is used as a control machine where no interaction is happening. On both virtual machines, four memory dumps are created while the virtual machine is paused. In between taking the four memory dumps on the non-control virtual machine, the execution of the tested tool is prepared and a memory dump created with it in between the third and fourth paused memory dump. This procedure is repeated ten times for each tool. The results show that Windows Memory Reader and DumpIt both have almost no impact on the system. In contrast, the usage of WinPmem and FTK Imager results in a difference of around 20% between the control and experimental virtual machine memory. The accuracy of the four tools is tested by comparing the paused memory dump taken before the acquisition using the tested tool with the memory dump produced by it. The memory dumps created with Windows Memory Reader and DumpIt reach about 80% similarity between the two memory dumps, followed by WinPmem at about 80% and lastly FTK Imager at below 80%.

Gruhn and Freiling [14] tested the causal consistency and integrity, see Section 2.4, of various memory acquisition techniques including kernel-level acquisition tools. They evaluated four kernel-level memory acquisition tools with a black box testing approach, i.e., without relying on changes to the source code of the tools. By writing known patterns in the memory of the virtual machine they can estimate the causal consistency and integrity of the acquired memory dumps. The tested kernel-level tools are FTK Imager, DumpIt, win64dd, and WinPmem. As a result, FTK Imager and WinPmem are almost at the same level regarding causal consistency and integrity. The same applies to DumpIt and win64dd. All kernel-level tools achieve less causal consistency and less integrity than other acquisition methods, for example, virtualization, cold boot and user-mode tools.

Kiperberg. et al. [18] proposed a hypervisor-based approach to main memory acquisition to avoid inconsistencies in a Windows 10 operating system. However, an evaluation of the effectiveness of the approach regarding inconsistencies is not discussed in this publication.

6.2 Measuring Causal Consistency

Pagani et al. [25] introduced the temporal aspect of a memory snapshot and called it *time consistency*. Time consistency describes whether the acquired physical pages by a tool match the actual physical pages at any point in time. They conducted an experiment to measure the impact of time consistency on memory snapshots by introducing three categories of inconsistencies, namely fragment inconsistency, pointer inconsistency and value inconsistency. As a result, only one memory snapshot out of ten was consistent and complete.

Ottmann et al. [24] conducted an experiment to investigate causal inconsistencies in memory dumps of Linux operating systems. With their approach the causal consistency of a part of the memory dump can be measured exactly. In their evaluation this part of the memory dump is limited to the heap of the so-called *pivot-program*. In the experiment, 360 memory dumps were generated. For each memory acquisition, system load was generated and the tested tool executed automatically. Subsequently, the memory dumps were analyzed regarding causal inconsistencies. This analysis was performed for the heap of the pivot program where vector clocks are used to track cause-effect relationships between elements of a list. The list elements serve as memory regions and events are accesses to the list elements by threads. Additionally, each memory dump was analyzed regarding inconsistencies in the VMA. As a result, a higher thread number seems to have an impact on the number of causal inconsistencies in the vector clocks. Additionally, a higher workload seems to influence the number of inconsistencies in the VMA.

7 Conclusion

In this work the occurrence of inconsistencies when acquiring main memory on Windows operating systems was analyzed. The VAD tree was used as an indicator of such inconsistencies. The number of nodes found in the VAD tree of each process was compared with a variable of the OS called `VadCount` which also contains the number of nodes of the tree. Additionally, a method to evaluate causal consistency was integrated in the experiments. The method which implements a way to search for causal inconsistencies was used before only in Linux operating systems. This work included the method for a Windows operating system.

In order to verify factors possibly influencing inconsistencies as workload, number of threads of the pivot program and execution time of the memory acquisition, three activity levels with different overall memory usage were defined. The experiments were conducted using an automation approach to generate a large dataset. The main memory was acquired with the tool WinPmem. As a result, the activity level seems to have a stronger impact on the execution time of WinPmem than the number of threads of the pivot program. With an increase of the activity level, more applications are started which run concurrently while acquiring the main memory. For this reason, WinPmem has to share more CPU time which likely causes a longer time to acquire the main memory. Additionally, both execution time and activity level seem to have an impact on the number of VAD inconsistencies. As most of the VAD inconsistencies were apparent in background processes of Windows, it is most likely that the pages allocated by these processes changed while acquiring the main memory. A longer execution time leads to a higher probability of pages changing, as well as a higher probability of encountering VAD inconsistencies in general. In contrast, the number of threads does not seem to have an impact on the number of VAD inconsistencies. In opposition to the assumption that there is an increase in the number of causal inconsistencies when using a higher number of threads as well as with the activity level, none of the named factors seems to have an impact on the number of causal inconsistencies when using one, two, and four threads for the pivot program. The relationship between the number of threads and the number of causal inconsistencies becomes only visible if at least eight threads are used for the pivot program. Likewise, the relationship between the number of causal inconsistencies and the activity level becomes visible when using a higher number of list elements in the pivot program.

In conclusion, the work showed that the method to track causal inconsistencies on Linux operating systems can also be applied to Windows operating systems. However, certain parameters need to be adjusted to the experiment environment, e.g., the number of threads. Additionally, this paper verified that the VAD comparison is a suitable indicator for inconsistencies in memory dumps. Furthermore, this work confirmed that inconsistencies are very likely to appear during the main memory acquisition of a Windows operating system. Therefore, as already suggested by Pagani et al. [25], an acquisition order should be defined, starting with the highest priority processes before acquiring the whole main memory. Additionally, a concurrently running memory acquisition could be adjusted to achieve a copy which looks like the machine was paused. This is called *quasi-instantaneous* by Ottmann et al. [24].

In a future work, redundant information in data structures on Windows operating systems can be researched in order to verify whether these are suitable as a consistency indicator. The found consistency indicators can then be compared with regard to reliability and influencing factors. Additionally, the indicators can be compared with consistency indicators of other operating systems. Furthermore, an experiment where the pivot program, which tracks causal inconsistencies, is started multiple times can be conducted to verify whether this leads to more inconsistencies. An experiment with more CPU capacity for the virtual machine can show whether the number of CPUs influences the execution time of the memory acquisition and therefore the probability of encountering inconsistencies, as well. Moreover, consequences of the occurrence of inconsistencies should be analyzed in order to find a more precise definition of the term “inconsistency.” Additionally, we can evaluate popular memory acquisition tools with our method to show whether inconsistencies affect the retrievability of certain artifacts, e.g., encryption keys, from memory. Lastly, this work showed that there is a need for a memory acquisition

tool which copies the memory regions without producing inconsistencies. Live virtualization and copy-on-write solutions are potential approaches to achieve this.

Appendix

A Statistical Tests

This section briefly describes the statistical test used throughout this work. The process of choosing the correct statistical test is for each analysis the following:

At first, a *Shapiro–Wilk normality test* is used to decide on the correlation method [8, Chapter 8]. The resulting *p-value* indicates the probability that the results have happened by chance [8, p. 347]. If the test yields a p-value smaller than 0.05, the residuals are not following a normal distribution. Therefore, a *Spearman correlation* needs to be conducted [27]. The resulting *R-value* indicates the effect size of the relationship [5]. Additionally, a *Levene's test* for homogeneity of variance of the residuals is conducted [21]. If the test yields a p-value smaller than 0.05, there is no homoscedasticity, i.e., the spread of the data are different for each group. If the residuals are neither following a normal distribution nor the variance is homogeneous, a *Kruskal–Wallis test* is conducted to determine if there are differences between the defined groups [32]. In particular, the test shows whether the groups are from the same population. If the p-value is smaller than 0.05, there are differences between the groups. As the Kruskal–Wallis test does not provide information about between which groups the differences are, a *post hoc* test needs to be conducted. Throughout this work, the residuals are not normally distributed. For this reason, a *Wilcoxon rank sum test* with Bonferroni correction is chosen as a *post hoc* test [17]. If the p-value is smaller than 0.05, the groups are significantly distinct from each other.

References

[1] Eric Barnes. 2021. Mitigating the risks of fileless attacks. *Computer Fraud & Security* 2021, 4 (2021), 20–20. DOI: https://doi.org/10.1016/S1361-3723(21)00044-0

[2] W. Campbell. 2014. Volatile memory acquisition tools - A comparison across taint and correctness. In *Proceedings of the 11th Australian Digital Forensics Conference (ADF '13)*, 10–19.

[3] Brian Carrier. 2005. *File System Forensic Analysis*. Addison-Wesley Professional.

[4] Andrew Case and Golden G. Richard III. 2017. Memory forensics: The path forward. *Digital Investigation* 20 (2017), 23–33. DOI: https://doi.org/10.1016/j.diin.2016.12.004

[5] Jacob Cohen and Jacob Willem Cohen. 1988. *Statistical Power Analysis for the Behavioral Sciences* (2nd ed.). Erlbaum, Hillsdale, NJ [u.a.]. XXI, 567 S. pages. Literaturverz. S. 553–558.

[6] Microsoft Corporation. 2023. Device Encryption in Windows. Retrieved March 20, 2023 from https://support.microsoft.com/en-us/windows/device-encryption-in-windows-ad5dcf4b-dbe0-2331-228f-7925c2a3012d

[7] Microsoft Corporation. 2023. RTL_AVL_TABLE structure (ntddk.h). Retrieved July 02, 2023 from https://learn.microsoft.com/en-us/windows-hardware/drivers/ddi/ntddk/ns-ntddk-_rtl_avl_table

[8] Michael J. Crawley. 2007. *The R Book* (1st. ed.). Wiley Publishing.

[9] Brendan Dolan-Gavitt. 2007. The VAD tree: A process-eye view of physical memory. *Digital Investigation* 4 (2007), 62–64. DOI: https://doi.org/10.1016/j.diin.2007.06.008

[10] Exterro. 2023. FTK® Imager. Retrieved April 25, 2023 from https://www.exterro.com/ftk-imager

[11] FireEye. 2023. Memoryze™. Retrieved April 25, 2023 from https://fireeye.market/apps/211368

[12] Magnet Forensics. 2023. MAGNET DumpIt. Retrieved April 25, 2023 from https://www.magnetforensics.com/resources/magnet-dumpit-for-windows/

[13] Volatility Foundation. [n. d.]. Changes between Volatility 2 and Volatility 3. Retrieved March 17, 2023 from https://volatility3.readthedocs.io/en/latest/vol2to3.html

[14] Michael Gruhn and Felix C. Freiling. 2016. Evaluating atomicity, and integrity of correct memory acquisition methods. *Digital Investigation* 16 (2016), S1–S10. DOI: https://doi.org/10.1016/j.diin.2016.01.003 DFRWS 2016 Europe.

[15] Apple Inc. 2023. Use FileVault to Encrypt Your Mac Startup Disk. Retrieved April 25, 2023 from https://support.apple.com/en-us/HT204837

[16] Hajime Inoue, Frank Adelstein, and Robert A. Joyce. 2011. Visualization in testing a volatile memory forensic tool. *Digital Investigation* 8 (2011), 42–51.

[17] Alboukadel Kassambara. 2024. Wilcoxon Test in R. Retrieved March 04, 2024 from https://www.datanovia.com/en/lessons/wilcoxon-test-in-r/

[18] Michael Kiperberg., Roee Leon., Amit Resh., Asaf Algawi., and Nezer Zaidenberg. 2019. Hypervisor-assisted atomic memory acquisition in modern systems. In *Proceedings of the 5th International Conference on Information Systems Security and Privacy (ICISSP '19)*. INSTICC, SciTePress, 155–162. DOI: https://doi.org/10.5220/0007566101550162
[19] Vidstrom Labs. 2023. PMDump. Retrieved April 25, 2023 from https://vidstromlabs.com/freetools/pmdump/
[20] Tobias Latzo, Ralph Palutke, and Felix Freiling. 2019. A universal taxonomy and survey of forensic memory acquisition techniques. *Digital Investigation* 28 (2019), 56–69. DOI: https://doi.org/10.1016/j.diin.2019.01.001
[21] Haiko Lüpsen. 2023. Varianzanalysen – Prüfen der Voraussetzungen und nichtparametrische Methoden sowie praktische Anwendungen mit R und SPSS. Retrieved October 15, 2023 from https://kups.ub.uni-koeln.de/6851/1/nonpar-anova.pdf
[22] Friedemann Mattern. 1988. Virtual time and global states of distributed systems. In *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, 215–226.
[23] Jenny Ottmann, Frank Breitinger, and Felix Freiling. 2022. Defining atomicity (and integrity) for snapshots of storage in forensic computing. In *Proceedings of the Digital Forensics Research Conference Europe (DFRWS EU '22)*, 11 pages, (2022-03-29/2022-04-01). Retrieved from https://dfrws.org/presentation/defining-atomicity-and-integrity-for-snapshots-of-storage-in-forensic-computing/
[24] Jenny Ottmann, Frank Breitinger, and Felix Freiling. 2023. An experimental assessment of inconsistencies in memory forensics. *ACM Transactions on Privacy and Security* 27, 1 (Dec. 2023), Article 2, 29 pages. DOI: https://doi.org/10.1145/3628600
[25] Fabio Pagani, Oleksii Fedorov, and Davide Balzarotti. 2019. Introducing the temporal dimension to memory forensics. *ACM Transactions on Privacy and Security* 22 (Mar. 2019), 1–21. DOI: https://doi.org/10.1145/3310355
[26] Bradley Schatz. 2007. BodySnatcher: Towards reliable volatile memory acquisition by software. *Digital Investigation* 4 (2007), 126–134. DOI: https://doi.org/10.1016/j.diin.2007.06.009
[27] Sereno. 2023. Comparison of Pearson vs Spearman Correlation Coefficients. Retrieved October 15, 2023 from https://www.analyticsvidhya.com/blog/2021/03/comparison-of-pearson-and-spearman-correlation-coefficients/
[28] StatCounter. 2024. Desktop Operating System Market Share Worldwide. Retrieved March 11, 2024 from https://gs.statcounter.com/os-market-share/desktop/worldwide
[29] Velocidex. 2023. WinPmem. Retrieved April 25, 2023 from https://github.com/Velocidex/WinPmem/
[30] Stefan Vömel and Felix Freiling. 2012. Correctness, atomicity, and integrity: Defining criteria for forensically-sound memory acquisition. *Digital Investigation* 9 (Nov. 2012), 125–137. DOI: https://doi.org/10.1016/j.diin.2012.04.005
[31] Stefan Vömel and Felix C. Freiling. 2011. A survey of main memory acquisition and analysis techniques for the windows operating system. *Digital Investigation* 8, 1 (2011), 3–22. DOI: https://doi.org/10.1016/j.diin.2011.06.002
[32] Björn Walther. 2023. Kruskal-Wallis Test in R rechnen. Retrieved September 21, 2023 from https://bjoernwalther.com/kruskal-wallis-test-in-r-rechnen/
[33] Pavel Yosifovich, David A Solomon, and Alex Ionescu. 2017. *Windows Internals, Part 1: System Architecture, Processes, Threads, Memory Management, and More*. Microsoft Press.

Received 19 April 2024; revised 19 April 2024; accepted 3 July 2024