Journal of Information Security and Applications 98 (2026) 104393

Contents lists available at ScienceDirect

Journal of Information Security and Applications

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

Accelerating volatile memory forensics for bare-metal malware analysis with FPGA devices

Dan Cristian Turicu, Florin Oniga *

Computer Science Department, Technical University of Cluj-Napoca, Cluj-Napoca, Romania

ARTICLE INFO

Keywords:
Volatile memory forensics
Bare-metal malware execution
FPGA-Based hardware acceleration
Windows 10 process monitoring

ABSTRACT

Modern malware often employs anti-analysis techniques to detect virtualized or emulated environments, evading traditional dynamic analysis systems. To address this challenge, bare-metal analysis platforms have emerged as a more transparent alternative. However, efficiently monitoring them while preserving transparency and minimizing interference remains a key challenge. In this paper, we present a proof-of-concept hardware accelerator implemented on an FPGA device, designed for high-speed volatile memory acquisition and on-the-fly pool tag scanning of the memory content to extract information about active and terminated processes on a bare-metal malware execution system running Windows 10. The memory forensics accelerator leverages PCIe-based DMA to acquire the volatile memory from the monitored system and performs the scanning for process structures directly on the FPGA, without requiring any software installation on the monitored system. Our approach improves transparency and isolation, and shows significant speed advantages over conventional snapshot-based memory forensics. We evaluate the prototype and discuss its limitations and applicability in malware analysis workflows.

1. Introduction

The rapid increase in the number, complexity, and variety of malicious applications has required the adoption of automated analysis techniques. Among these, automated dynamic malware analysis has become a widely used approach, relying on sandbox environments to execute and observe the potentially malicious actions of analyzed software samples. In practice, such environments commonly employ virtualization or emulation, executing malware within virtual machines managed by a hypervisor or emulated operating systems. These controlled and monitored environments facilitate the extraction of behavioral patterns and harmful actions by isolating the malware from the host system. Additionally, they provide the advantage of quickly restoring the environment to a clean state after each sample is analyzed.

However, virtualization and emulation technologies inevitably introduce artifacts into the analysis environment, which can be detected by modern malware. Evasive or stealthy malware often incorporates anti-analysis capabilities that attempt to identify such environments [1,2] by exploiting artifacts introduced by the emulation layer [3–6] or the virtualization infrastructure [7,8], thereby evading the analysis process.

To overcome this limitation, researchers and practitioners have turned to bare-metal malware analysis systems, in which the operating system executes directly on physical hardware without intermediary layers. By eliminating artifacts introduced by emulation or virtualization, such systems are less detectable to sophisticated malware. For effective dynamic analysis, an ideal bare-metal platform must support comprehensive monitoring capabilities, including access to system memory, visibility into the storage device modifications, and inspection of network traffic generated during sample execution.

In this paper, we focus on the volatile memory of a bare-metal malware analysis system. We present a proof-of-concept implementation of a high-speed volatile memory acquisition method on an FPGA device, together with an efficient technique for extracting the list of active and terminated processes. The proposed solution demonstrates the feasibility and potential performance benefits of hardware-assisted memory forensics on bare-metal systems, i.e., systems in which software executes directly on physical hardware without relying on virtualization, emulation, or a hypervisor layer.

The current implementation is tailored to a specific hardware configuration and assumes a controlled laboratory environment where virtualization support is disabled. For this reason, it may not be directly applicable to production systems where virtualization or additional security mechanisms are active. Although more invasive approaches that directly instrument the memory subsystem are possible, our approach focuses on a practical, immediately deployable solution suitable for adoption on existing commodity systems used for bare-metal malware analysis, where minimal platform modification, ease of integration, and rapid deployment are essential requirements.

* Corresponding author.
E-mail addresses: dan.turicu@cs.utcluj.ro (D.C. Turicu), florin.oniga@cs.utcluj.ro (F. Oniga).

https://doi.org/10.1016/j.jisa.2026.104393

Available online 10 February 2026

The predominant approach in both academia and digital forensic investigations involves the acquisition of volatile memory snapshots using specialized software-based [9,10] or hardware-based solutions [11–14], typically at the beginning and end of sample execution. These snapshots are subsequently analyzed using memory forensics frameworks such as Volatility [15] and Rekall [16], either by executing a differential comparison between the two snapshots or by performing a full analysis of each snapshot individually. In contrast, the proposed solution combines volatile memory acquisition with on-the-fly forensic artifact scanning, enabling the extraction of forensic artifacts directly during memory acquisition rather than relying on separate analysis of memory snapshots.

While the solution described in this paper focuses on volatile memory acquisition and forensic artifact scanning, effective malware analysis requires integrating this capability with a mechanism that restores the analysis environment to a clean state after each sample execution, such as the storage mirroring technique described in our previous work [17], which enables restoration of the bare-metal system's storage device.

To the best of our knowledge, no prior work combines volatile memory acquisition with real-time scanning of forensic artifacts using an FPGA device. The main contributions of this paper are:

1. The design of a hardware architecture for high-speed volatile memory acquisition and real-time forensic artifact scanning on FPGAs;
2. An efficient FPGA-based method for detecting active and terminated processes on Windows 10 through direct physical memory scanning;
3. A proof-of-concept evaluation on a bare-metal system demonstrating significant performance improvements and reduced overhead compared to traditional memory forensics tools;
4. An assessment of the method's limitations and applicability in both laboratory and real-world environments.

2. Theoretical background

2.1. Pool tag scanning for process detection

In the domain of digital forensics and memory analysis, the accurate identification of active and terminated processes is important for understanding the state of a system at a given point in time. By examining these processes, investigators can reconstruct the sequence of events that occurred on a system, including when programs were executed.

Traditional techniques for enumerating active processes rely on traversing the linked lists maintained by the operating system. However, these approaches are susceptible to anti-forensic techniques, such as rootkits that unlink malicious processes by manipulating the operating system's internal data structures to evade detection.

To overcome these limitations, pool tag scanning provides a more resilient method for discovering executive objects, i.e., kernel-mode objects managed by the operating system, that operates independently of potentially compromised operating system level structures. This technique involves scanning volatile memory for specific allocation patterns and object signatures associated with kernel pool allocations.

In Windows operating systems, kernel pools are memory regions where executive objects are dynamically allocated and managed. Processes, threads, and synchronization primitives are examples of executive objects residing within these memory pools.

By analyzing the kernel pool allocations, forensic tools can identify and extract executive objects, providing critical insights into the system's state and behavior. Each executive object allocation begins with a pool header structure, followed by several optional headers and an object header. The object header immediately precedes the executive object structure in memory, with the optional headers located before the object header in a fixed order, resulting in a predictable memory layout. Once a valid object header is located, determining the corresponding executive object structure is straightforward. Fig. 1 shows the memory layout for a process structure, along with the main fields of the structures used to identify the process.

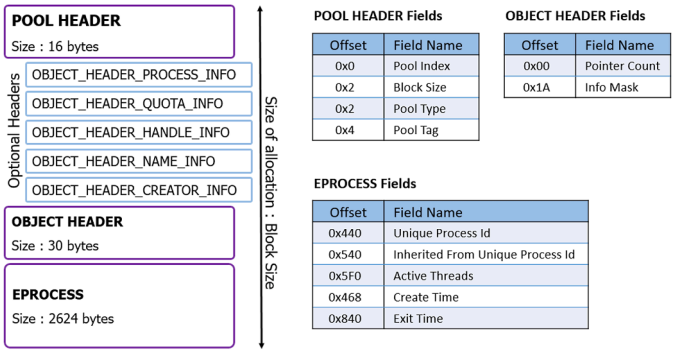

Fig. 1. Memory layout of a process executive object and the key fields.

The Volatility framework's *psscan* plugin employs this approach by searching for process structures directly within memory pools. By leveraging known characteristics of process objects, such as pool tags, size, and structural layout, *psscan* can identify active and terminated processes, including those that have been intentionally hidden.

Our memory forensics accelerator adopts a similar approach, enabling the efficient retrieval of active and terminated process information through direct scanning of the physical memory.

2.2. PCIe-based DMA memory acquisition

Direct Memory Access (DMA) over the PCI Express (PCIe) interface has emerged as a powerful technique for high-speed physical memory acquisition, particularly in forensic and malware analysis contexts. DMA allows a hardware device to access physical system memory independently of the CPU, enabling memory acquisition with minimal reliance on the target operating system–an important advantage when that system may be compromised or running evasive malware.

DMA-based memory acquisition techniques typically involve the use of PCI/PCIe cards that initiate and manage memory transfers directly. These devices issue memory read requests over the PCI/PCIe bus, bypassing standard operating system level access controls and achieving high throughput. This architecture makes PCIe-based DMA memory acquisition ideal for use in bare-metal malware analysis systems, where low-level visibility and minimal interference are critical.

Because this approach operates below the operating system level, it is inherently resistant to many anti-forensic techniques, including those that attempt to manipulate or hide memory regions from software-based acquisition tools. However, modern system security features, such as Intel VT-d (Virtualization Technology for Directed I/O) and IOMMU (Input-Output Memory Management Unit), can restrict DMA access to certain memory regions. As a result, for PCIe-based DMA acquisition to function properly, these protections must typically be disabled.

In controlled laboratory environments or dedicated bare-metal malware analysis systems, this limitation is acceptable and provides a high degree of visibility into system memory without modifying the state of the system being analyzed.

FPGA-based PCIe DMA engines also offer potential for real-time memory scanning, as memory content can be processed on-the-fly without the need to store full memory snapshots. Unlike traditional approaches that require full memory acquisition followed by offline analysis, these engines can stream volatile memory content directly to onboard logic for immediate processing. This enables the detection of artifacts during data transfer, significantly reducing the time to insight and minimizing storage requirements by eliminating the need to store large memory dumps.

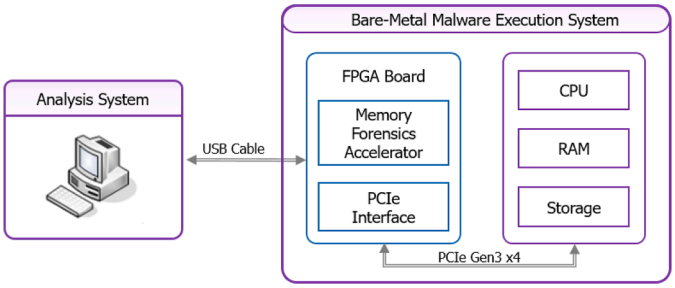

Fig. 2. System architecture.

3. Memory forensics accelerator

3.1. System overview

The memory forensics accelerator is implemented on an FPGA board equipped with a PCIe connector, which allows integration into the bare-metal malware execution system. Through the PCIe interface, the memory forensics accelerator reads the physical memory of the malware execution system and performs on-the-fly scanning of the volatile memory to detect forensic artifacts. The extracted artifacts, i.e., the list of active and terminated processes, are transmitted via a serial connection to the analysis machine. The architecture of the system is shown in Fig. 2.

By leveraging an FPGA board with a PCIe interface, the accelerator achieves its primary design objectives: *fast acquisition* of the bare-metal malware execution system's physical memory and *efficient scanning* of its content for specific forensic artifacts.

In addition to performance, the FPGA-based accelerator enables three key security features [18]: *complete view* of the volatile memory, *transparent deployment* of the accelerator, and *strong isolation* between the accelerator and the malware execution system. Transparency is achieved by eliminating the need to install any software components on the malware execution system. The only artifact introduced is the presence of the FPGA board, which can be obfuscated by changing its identification parameters and masquerading as a different device, thereby evading detection by malware samples that search for specific hardware signatures.

3.2. Embedded system architecture

The architecture of the memory forensics accelerator implemented on the FPGA, shown in Fig. 3, includes a *MicroBlaze Processor* and a custom *Memory Scanner* peripheral that implements the core functionality.

The processor initiates the physical memory acquisition and collects the detected artifacts. Once the acquisition completes, the processor performs light post-processing of the collected artifacts before transmitting the results to the analysis machine via the serial connection.

The core functionalities of the *Memory Scanner* peripheral are to acquire the physical memory of the bare-metal malware execution system and to detect forensic artifacts, specifically active and terminated processes, within the retrieved data. The peripheral exposes several registers that the processor uses to initiate memory acquisition and to monitor its status. During acquisition, the peripheral continuously scans the memory content for process-related artifacts. When a process is detected, the peripheral triggers an interrupt to notify the processor. The processor then retrieves the process details from the peripheral's internal memory. Upon completion of the acquisition and scanning process, an interrupt is issued to signal the end of the operation. The processor then reads the final status, processes the collected artifacts, transmits the results via the *AXI Uart Lite* component, and can subsequently initiate a new physical memory acquisition.

3.3. Memory scanner peripheral

The architecture of the *Memory Scanner* peripheral, shown in Fig. 4, includes the *PCIe Core*, the *Requester Engine*, the *Process Scanner*, and a set of registers used by the *MicroBlaze Processor* to control and read the status of the peripheral.

The *PCIe Core* component of the peripheral instantiates a hardened PCI Express core [19] featured in AMD UltraScale+ FPGA devices, which implements the PCIe interface functionality as a fixed, dedicated block. The integrated core supports PCIe Gen3 data transfer rates and was configured to use four lanes. For the memory forensics accelerator, only the *Requester Request* and *Requester Completion* user data interfaces of the PCI Express core were used, as the design requires only outbound transactions initiated by the FPGA to access the host system's physical memory. *Memory Read Request* transactions generated by the *Requester Engine* are sent through the *Requester Request* interface, while data read from memory are received through the *Requester Completion* interface.

The *Requester Engine* functions as a Bus Master DMA engine and generates PCIe transactions to read the host system's physical memory. In a given system, volatile memory is mapped to several physical address ranges, determined by the system's firmware and hardware configuration. These configurations define fixed regions for memory allocation.

During a memory acquisition, the *Requester Engine* initiates read operations for a complete physical memory page along with the subsequent physical page. This approach ensures that process structures starting near the end of a physical page and extending into the next page are fully captured. After generating a read request, the engine waits until

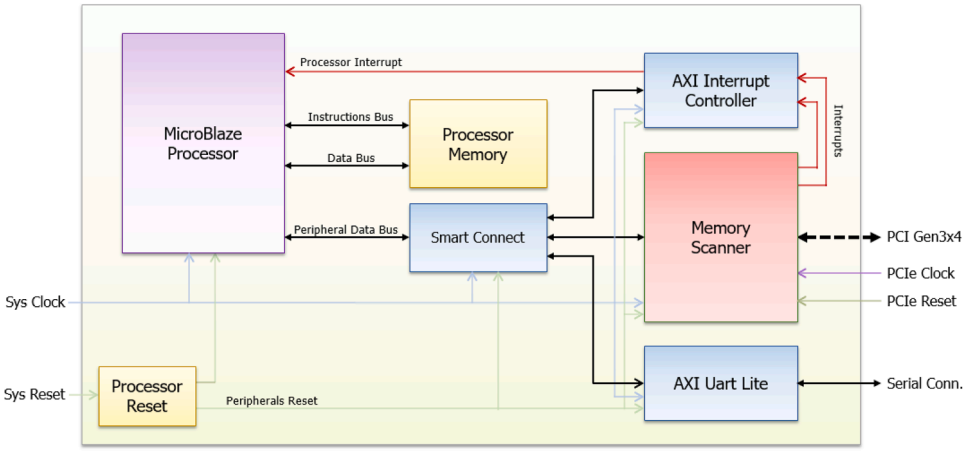

Fig. 3. Embedded system architecture implemented on the FPGA device.

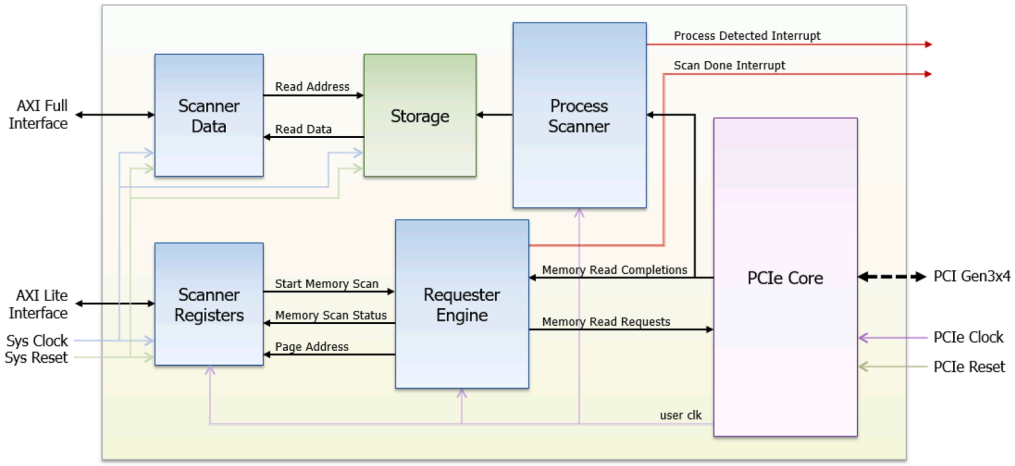

Fig. 4. Memory Scanner Peripheral architecture.

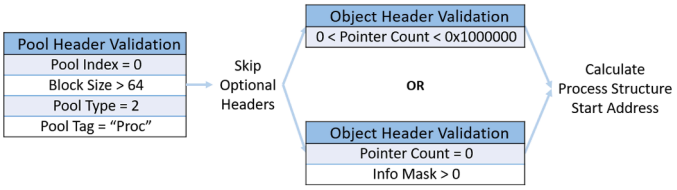

Fig. 5. Pool Headers and Object Headers conditions for a valid process.

the content of both pages is received and then it proceeds by requesting the next pair of pages. Although this results in each physical page being read twice, it simplifies the engine's control logic and increases the likelihood that pages are received in the order in which they were requested. When all the physical pages of the volatile memory are read, the engine asserts the *Scan Done Interrupt* to notify the processor that the memory acquisition is complete.

The *Process Scanner* receives and scans the physical memory pages requested by the *Requester Engine*. It first checks whether the data contains a valid pool header associated with a process. Once a pool header passes all implemented validation checks, the component searches for a corresponding object header. If a valid object header follows the pool header, the bytes of the process structure are saved to the *Storage* memory. When all process structure bytes have been stored, a *Process Detected Interrupt* is triggered to notify the processor that a valid process has been saved and that it can now read the fields of interest from the *Storage*.

The checks performed by the *Process Scanner* component are inspired by the *psscan* plugin of the Volatility framework. Fig. 5 illustrates the implemented validations for the pool header corresponding to a process executive object, as well as the validations for the object header.

Process detection begins by checking all conditions associated with the pool header. If these conditions are satisfied, the optional headers are skipped, and then the object header conditions are evaluated. If at least one of the object header conditions is met, the *Process Scanner* determines the start of the process structure in the incoming data and once the corresponding memory data are received, the bytes are stored in the *Storage*.

The processor uses the *Scanner Registers* to control and read the status of the *Requester Engine*. The processor initiates a memory acquisition by setting a specific bit in the Control Register. The Status Register contains flags that indicate the completion of memory acquisition, as well as an error bit that is asserted if errors occurred during the memory acquisition process. Additionally, the registers set includes the Page Address Register, which is updated with the physical address of the memory page each time a new process is detected.

The *Scanner Data* component provides an interface for the processor to retrieve details of detected processes. Once the *Process Scanner* saves a newly identified process into the *Storage* memory, an interrupt is triggered to notify the processor, which then uses the *Scanner Data* interface to read the relevant fields of the detected process.

3.4. Microblaze software application

The software application running on the *MicroBlaze Processor* performs several key tasks: it initializes the interrupt controller, initiates a memory acquisition by writing to the Control Register of the *Memory Scanner* peripheral, and handles interrupts throughout the scanning process. When the *Memory Scanner* detects a new process and asserts the *Process Detected Interrupt*, the processor reads the physical address of the memory page containing the process from the *Scanner Registers*, along with the relevant process fields from the *Storage* memory. These results are temporarily stored for further processing. Upon completion of the memory acquisition, signaled by the *Scan Done Interrupt*, the processor performs light post-processing tasks, such as converting time-related fields, and transmits the results to the analysis machine via the serial connection.

3.5. Threat model

Our system is designed for use in controlled laboratory environments operated by security companies or research institutions, where large-scale malware analysis is conducted. In this context, our approach complements existing virtualized infrastructures used for dynamic malware analysis by providing a transparent bare-metal alternative that can operate in parallel with conventional sandbox-based systems. The objective is to enable the observation and analysis of malware behavior on real hardware while maintaining strong resistance to tampering and evasion.

We assume that the bare-metal malware execution system may be fully compromised, including the presence of kernel-level malicious code. Because our design operates entirely out-of-band, the acquisition process does not depend on the integrity of the host operating system. It relies solely on the FPGA device to perform direct memory access (DMA) operations, and no software components are required on the malware execution system.

We assume that the malware under analysis is not explicitly designed to thwart semantic-gap reconstruction techniques, such as by dynamically altering key operating system structures used for interpretation of physical memory. The focus of this work is on ensuring transparent, tamper-resistant memory acquisition and artifact scanning, not on countering advanced anti-forensic techniques that intentionally disrupt semantic analysis.

Currently, the evaluation does not include malware specifically designed to detect hardware monitoring devices or unusual PCIe activity. Our primary objective is to present a novel hardware technique that combines volatile memory acquisition with forensic artifact scanning. Validation against evasive malware will be conducted in future work after integrating this approach with the system restoration mechanism described in [17]. Nonetheless, the architectural transparency of our design, i.e., no software components on the monitored system and the ability to masquerade the FPGA as different PCIe devices, provides a strong foundation for resisting detection by malware.

3.6. Limitations

The current approach relies on direct access to the system's physical memory address space, which is incompatible with Intel Virtualization Technology for Directed I/O (VT-d). This is a hardware-assisted feature that provides memory isolation and access control by remapping Direct Memory Access (DMA) requests from peripheral devices. While this enhances system security and stability by preventing unauthorized memory access, it also restricts the memory forensics accelerator from performing low-level memory acquisition. As a result, VT-d feature must be disabled for the accelerator to function correctly.

Another limitation of the proposed approach is related to modern memory encryption mechanisms, such as Intel Total Memory Encryption (TME) or AMD Secure Memory Encryption (SME). These technologies encrypt the contents of system memory using keys managed internally by the processor, preventing external devices from interpreting data obtained through DMA. As a result, our hardware accelerator would not be able to perform memory acquisition and artifact scanning in such environments. Similar to the requirement of disabling VT-d to allow unrestricted DMA access, the absence of memory encryption support is a necessary condition for the correct operation of the proposed system.

In this research, the deactivation of VT-d and memory encryption mechanisms does not represent a limitation, as the system is a dedicated bare-metal malware execution system operating in a controlled laboratory environment. This setup is not intended for general-purpose use, where such features would normally be required to ensure system integrity and data confidentiality. In our case, the primary objective is to enable memory forensics and analysis without interference from hardware-level access restrictions. Therefore, disabling VT-d and memory encryption is an acceptable and necessary configuration choice to facilitate effective accelerator deployment and evaluation.

3.7. Future enhancements

As a proof-of-concept, the current implementation accelerates volatile memory acquisition and scanning and is limited to detecting active and terminated processes. Future possible enhancements can introduce additional components (similar to the *Process Scanner*) which perform parallel scans of memory pages to identify other types of executive objects, such as threads, synchronization primitives, registry keys.

Another enhancement for a more production-oriented environment involves replacing the serial connection with an Ethernet interface. This modification would enable the analysis machine to manage and collect results from multiple bare-metal systems running malware samples concurrently, all connected to a local area network.

4. Experimental results

The AMD Vivado Design Suite 2023.2 [20] was used to design, implement, test, and evaluate the proposed memory forensics accelerator on an AMD Alveo U50 Data Center Accelerator Card [21], which features an UltraScale+ FPGA device. The FPGA board was integrated into a Dell Precision 5820 workstation equipped with an Intel Xeon W-2123 processor and 32 GB of DDR4 RAM (4 × 8 GB, 2666 MHz).

Table 1
Physical memory address ranges on the Dell Precision 5820.

Start Address	End Address	Size	Pages
0x1000	0x3F000	248 KiB	62
0x40000	0xA0000	384 KiB	96
0x100000	0x49CBD000	1,208,052 KiB	302,013
0x4CBBA000	0x4E700000	27,928 KiB	6982
0x1000000000	0x8A00000000	31,981,568 KiB	7,995,392
		33218180 KiB	**8304545**

The test system, running Windows 10 Pro, version 22H2 (build 19045), simulates a bare-metal malware execution system. Table 1 shows the physical address ranges to which the volatile memory is mapped on this specific system. These ranges were identified using *RAMMap* [22], a low-level diagnostic tool, and are determined by the machine's firmware and memory controller configuration.

4.1. Results validation

To validate the results generated by the memory forensics accelerator, we performed a volatile memory acquisition and forensic artifact scan using the accelerator. Subsequently, a snapshot was acquired with *WinPmem* [9], processed through the Volatility framework, and the results were compared.

Once Windows 10 had fully loaded, we launched several random applications to simulate the behavior of a typically used system, after which we performed the following sequence of operations. First, we performed a complete memory acquisition and scan using the memory forensics accelerator. Once the list of active and terminated processes was obtained, we acquired a memory snapshot using *WinPmem*. We then saved a *Process Explorer* [23] report for later reference. Finally, we analyzed the snapshot using the *psscan* plugin of Volatility Framework [15] and compared the results.

The results obtained during a validation round were as follows: the memory forensics accelerator detected 247 processes, while the Volatility framework reported 245. A comparison of the two lists revealed eight discrepancies, with the remaining processes matching. Meanwhile, the *Process Explorer* report listed only 167 active processes. This difference is expected, as *Process Explorer* shows only active processes, whereas both the memory forensics accelerator and Volatility also report recently terminated ones.

Apart from the 8 differing processes, the only discrepancies between the two tools for the remaining entries were in the reported number of threads per process; all other process details matched exactly. Fig. 6 shows several selected examples comparing the outputs of the Volatility *psscan* plugin with the corresponding results reported by the memory forensics accelerator. We repeated this sequence of operations across multiple validation rounds, consistently observing similar results.

4.2. Performance evaluation

To evaluate the performance improvement, we compared the time required to perform a full memory analysis using the tools. A complete volatile memory acquisition and scanning using the memory forensics accelerator on the test system takes approximately 26 seconds. This duration reflects the time required to acquire volatile memory, with scanning performed on-the-fly as data are received. This eliminates the need for local storage to save full memory snapshots.

To measure the read throughput of the forensics accelerator, we used the *Intel Performance Counter Monitor (PCM)* during a full physical memory acquisition. The *pcm-iio* tool recorded the number of bytes read per second at one-second intervals, reporting an average throughput of 2854 MB/s. This value represents the effective data transfer rate on the PCIe bus and does not account for the fact that each memory page is read twice during the acquisition process.

Volatility Generated Output

```
Volatility 3 Framework 2.26.0
Progress:  100.00               PDB scanning finished
PID     PPID    ImageFileName   Offset(P)       Threads Handles Wow64   CreateTime                      ExitTime
4       0       System          0x89a294040     222     -       False   2025-05-21 15:12:10.000000 UTC  N/A
124     4       Registry        0x89a3b3040     4       -       False   2025-05-21 15:12:08.000000 UTC  N/A
11328   668     ApplicationFra  0x85b727080     4       -       False   2025-05-30 07:48:32.000000 UTC  N/A
11352   10472   python.exe      0x85b72f080     0       -       False   2025-05-26 12:40:20.000000 UTC  2025-05-26 12:42:21.000000 UTC
5748    612     Code.exe        0x831ff92c0     0       -       False   2025-05-21 15:20:53.000000 UTC  2025-05-21 15:31:15.000000 UTC
```

Memory Forensics Accelerator Output

```
Physical Page   : 0x831FF9000

Process ID      :       5748
Parent  ID      :        612
Active Threads  :          0
Image File Name : Code.exe

Create time     : 2025-05-21 15:20:53 UTC
Exit time       : 2025-05-21 15:31:15 UTC

Physical Page   : 0x85B727000

Process ID      :      11328
Parent  ID      :        668
Active Threads  :          6
Image File Name : ApplicationFra

Create time     : 2025-05-30 07:48:32 UTC
Exit time       : Running

Physical Page   : 0x85B72F000

Process ID      :      11352
Parent  ID      :      10472
Active Threads  :          0
Image File Name : python.exe

Create time     : 2025-05-26 12:40:20 UTC
Exit time       : 2025-05-26 12:42:21 UTC

Physical Page   : 0x89A294000

Process ID      :          4
Parent  ID      :          0
Active Threads  :        222
Image File Name : System

Create time     : 2025-05-21 15:12:10 UTC
Exit time       : Running

Physical Page   : 0x89A3B3000

Process ID      :        124
Parent  ID      :          4
Active Threads  :          4
Image File Name : Registry

Create time     : 2025-05-21 15:12:08 UTC
Exit time       : Running
```

Fig. 6. Processes detected by Volatility's *psscan* and the memory forensics accelerator.

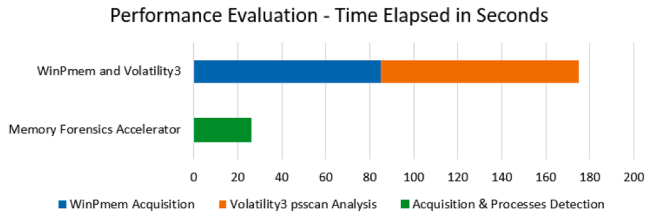

Fig. 7. Elapsed time comparison for a memory analysis.

A similar evaluation using Volatility involves measuring the time required to acquire the memory snapshot plus the time needed to analyze it. The test system's memory acquisition using *WinPmem* takes approximately 85 seconds when using an SSD, while processing the snapshot with *psscan* takes about 90 seconds. This does not include the additional time needed to download the symbol files (PDBs) for the specific operating system. Also, the Volatility use case does not take into account any additional time needed for transferring the memory snapshot to the analysis machine. Fig. 7 shows that the memory forensics accelerator reduces the acquisition and analysis time by a factor of approximately 6.5 times, from 175 to 26 seconds.

Our approach performs a linear scan of the entire physical memory, resulting in the scanning of 8,304,545 pages on the test system. In contrast, the *psscan* plugin performs an optimized scan across areas of physical memory that correspond to the kernel's memory allocations, scanning only about 410,690 pages of a memory snapshot acquired from the same test system. Consequently, the time required by the memory forensics accelerator to complete a memory scan could be significantly reduced by limiting the number of scanned pages.

The performance of the memory accelerator can be further improved by configuring the *PCIe Core* to operate in Gen3 x8 mode rather than the current Gen3 x4 configuration. Doubling the PCIe bandwidth significantly reduces the scan time, cutting it roughly in half compared to the current setup and improving overall data transfer efficiency.

4.3. Performance impact on memory bandwidth

To evaluate the impact on the effective memory bandwidth, we used the *RAMSpeed* [24] and *GFX Memory Speed Benchmark* [25].

Table 2
RAMSpeed Light Memory Load.

Test Iteration	Baseline MiB/s	Accelerator MiB/s	Delta %
INTmem 1	13,271	13,274	
INTmem 2	13,358	13,256	
INTmem 3	13,180	13,156	**-0.92**
INTmem 4	13,522	13,254	
INTmem 5	13,579	13,354	
FLOATmem 1	14,209	14,161	
FLOATmem 2	14,357	14,342	
FLOATmem 3	14,425	14,093	**-0.90**
FLOATmem 4	14,306	14,115	
FLOATmem 5	14,136	14,079	
MMXmem 1	14,398	14,175	
MMXmem 2	14,349	14,120	
MMXmem 3	14,268	14,156	**-1.31**
MMXmem 4	14,322	14,069	
MMXmem 5	14,291	14,169	

RAMSpeed benchmark provides memory tests based on the INT, FLOAT, and MMX instruction sets. Each test comprises four subtests (copy, scale, add, and triad), which collectively simulate typical memory-intensive operations encountered on the tested system. The memory tests were executed multiple times to evaluate performance impact. For each iteration, the memory test was first run with the memory forensics accelerator disabled to establish a baseline. The same test was then repeated while the accelerator was actively performing a complete memory acquisition and scan. In both cases, the four subtests were executed three times, and the reported values in the tables represent the average results.

The *RAMSpeed* measurements are shown in Table 2 and Table 3, where each row represents a test iteration. Performance penalties, calculated for each instruction set, are shown in the final columns. The evaluation in Table 2 measured the performance impact under light memory load conditions, while the system was running multiple applications with relatively low memory usage.

The second evaluation measured the impact on memory throughput under medium memory load conditions. In this case, the system was

Table 3
RAMSpeed Medium Memory Load.

Test Iteration	Baseline MiB/s	Accelerator MiB/s	Delta %
INTmem 1	12,764	12,732	
INTmem 2	12,758	12,669	
INTmem 3	12,838	12,685	**-0.68**
INTmem 4	12,797	12,732	
INTmem 5	12,803	12,704	
FLOATmem 1	13,599	13,372	
FLOATmem 2	13,544	13,388	
FLOATmem 3	13,462	13,400	**-1.19**
FLOATmem 4	13,553	13,408	
FLOATmem 5	13,615	13,397	
MMXmem 1	13,577	13,419	
MMXmem 2	13,531	13,351	
MMXmem 3	13,531	13,383	**-1.2**
MMXmem 4	13,459	13,403	
MMXmem 5	13,601	13,330	

Table 4
GFX Memory Speed Benchmark.

Test Iteration	Baseline MiB/s	Accelerator MiB/s	Delta %
Iteration 1	11,687	11,119	
Iteration 2	11,574	10,813	
Iteration 3	11,217	10,971	**-3.02**
Iteration 4	11,398	11,281	
Iteration 5	11,322	11,285	

running the same set of applications with relatively low memory usage, along with the *GFX Memory Speed Benchmark*, which generated high memory traffic. The benchmark was configured to continuously read data from memory throughout the execution of the test iterations. The results in Table 3 reflect the system's performance under these more demanding conditions.

Across all evaluations performed under varying memory load conditions, the maximum observed impact on memory throughput, as measured by *RAMSpeed*, remained below 2%.

GFX Memory Speed Benchmark is a performance testing tool designed to measure memory read and write speeds. It specifically evaluates how quickly the processor can access the system's memory, providing insight into overall memory subsystem performance. The benchmark was configured to allocate a total of 1 GB of memory. During each test iteration, the benchmark was first run to establish a baseline, and then repeated while the memory forensics accelerator was actively performing memory acquisition and scanning. A total of 50 memory read measurements were collected during each test run, and the average results are shown in Table 4. The observed impact on memory throughput was approximately 3% overall.

The measured results demonstrate an average impact of less than 5% on memory throughput when the memory forensics accelerator performs acquisition and scanning of the test system's physical memory. Taking into account the inherent variability introduced by operating system multitasking, multiple levels of caching, and differences between typical memory usage and benchmark execution, the memory forensics accelerator demonstrates a reduced impact on memory transfer rates.

4.4. FPGA device resource utilization

The programmable resources of the AMD UltraScale+ XCU50 FPGA, used for the implementation of the memory forensics accelerator, are summarized in Table 5. For each resource type, the number of available units is shown, while the corresponding table row indicates the utilized resources. The low utilization of logic resources leaves ample room for integrating additional modules and supporting future enhancements.

5. Related work

5.1. Memory forensics frameworks

Memory forensics frameworks such as *Volatility* [15] and *Rekall* [16], are widely used in both academic research and professional investigations for analyzing volatile memory snapshots. These tools operate by parsing the memory snapshots of a system to reconstruct high-level operating system structures, such as processes, threads, and network connections. They rely on knowledge of internal operating system structures, which are matched using signature-based or heuristic techniques. While powerful and extensible, these frameworks introduce performance overhead and analysis delays due to the acquisition of full memory snapshots. Our memory forensics accelerator mitigates these issues by integrating both memory acquisition and volatile memory processing within the FPGA device.

5.2. Memory acquisition techniques

Memory acquisition is a critical first step in volatile memory forensics, providing a snapshot of the system's state. Traditional methods rely on software-based tools such as *WinPmem* [9] and *DumpIt* [10], which are typically executed from within the target system's operating environment. These tools interface with kernel drivers or system APIs to read physical memory and save it to disk for offline analysis. While convenient and widely supported, these approaches may leave traces in memory and can be detected by sophisticated malware. Additionally, their performance is limited by storage device bandwidth, which can result in prolonged acquisition times.

To overcome these limitations, researchers and practitioners have explored DMA-based methods, which use hardware devices to access the physical memory directly from an external device. These methods leverage the capability of DMA devices to read physical memory without CPU involvement or operating system mediation, thereby enabling stealthy and tamper-resistant memory acquisition.

The effectiveness of using off-the-shelf [13,14] or specialized PCI/PCIe devices [11,12] for forensic memory acquisition has been well demonstrated, particularly in compromised environments. However, most DMA-based tools focus primarily on raw memory extraction and rely on subsequent offline analysis. Copilot [12] introduces a hardware-assisted kernel integrity monitor that uses a dedicated PCI-based coprocessor to verify the integrity of selected kernel memory regions against trusted baselines. While effective for detecting kernel-level compromises through out-of-band monitoring, Copilot is limited to integrity verification. In contrast to these approaches, our solution goes beyond memory acquisition and integrity monitoring by combining high-speed volatile memory acquisition with on-the-fly forensic artifact scanning. The design enables semantic reconstruction of runtime operating system structures during acquisition and reduces reliance on large storage and subsequent processing by integrating a DMA engine with on-device analysis capabilities.

A recent research direction explores the use of RDMA-capable NICs for memory acquisition in data-center environments [26]. This approach builds on the foundational concepts introduced by earlier systems [11–14], which leveraged DMA for external memory inspection, and adapts them to modern hardware through advanced RDMA features. In contrast to RDMA-based methods, which require driver installation and software configuration on the target host, our approach eliminates software dependencies on the monitored system, at the cost of disabling VT-d. This design choice enables a more transparent deployment model, making the proposed solution better suited for bare-metal malware analysis. In addition, our FPGA-based architecture

Table 5
Resource utilization on AMD UltraScale+ XCU50 FPGA.

UltraScale+ XCU50 FPGA	FF		LUT		BRAM		GT		PCIe	
	1743360		871680		1344		20		5	
Memory Forensics Accelerator	7394	0.42%	3585	0.41%	56	4.17%	4	20%	1	20%

supports on-device processing of the acquired volatile memory, enabling direct extraction of forensic artifacts during acquisition.

5.3. Bare-metal malware analysis platforms

The advantages of bare-metal environments for the analysis of evasive malware have been widely explored in academia. Bare-metal analysis platforms are dedicated physical systems designed to analyze malware in environments that closely resemble real-world conditions, without relying on virtualization or emulation. Unlike traditional sandboxing solutions that rely on virtual machines, bare-metal platforms run the operating system directly on physical hardware, eliminating artifacts detectable by evasive malware. This makes them particularly effective for analyzing sophisticated threats equipped with anti-analysis mechanisms.

These systems often include instrumentation for monitoring system activity while maintaining a high degree of transparency to the malware sample. However, since they lack the isolation and quick environment restoration capabilities of virtualized systems, bare-metal platforms require additional mechanisms to restore clean system states between analyses.

Several research efforts have addressed the challenges of executing malware samples on bare-metal platforms. One such FPGA-based bare-metal system described in the academic literature is LO-PHI [27], which uses an FPGA development board to monitor the memory and storage device activity of a bare-metal malware execution machine. For memory acquisition, LO-PHI leverages PCIe-based DMA to produce memory snapshots, which are subsequently transferred and post-processed using Volatility on a separate analysis machine. In contrast, our accelerator performs real-time processing of the acquired volatile memory, reducing the need for offline analysis.

BareBox [28] is a bare-metal malware analysis framework that provides rebootless system restoration capabilities by restoring both physical memory and storage device state using a custom operating system running on the analysis machine. BareCloud [29] further increases transparency by removing the monitoring module from the analysis machine and instead collecting storage and network activity at the hardware level. Rather than restoring the storage device locally, BareCloud leverages a remote storage system with copy-on-write techniques to reset device state. Compared to these systems, our approach offers a more transparent deployment model by relying entirely on out-of-band hardware-assisted memory acquisition and analysis.

6. Conclusion

This paper presents a hardware-assisted approach to volatile memory forensics tailored for bare-metal malware analysis systems. Leveraging an FPGA board with a PCIe interface, our memory forensics accelerator performs real-time memory acquisition and scanning for active and terminated processes using a pool tag scanning strategy inspired by Volatility. Unlike traditional forensic tools that depend on memory snapshots, our design enables software-independent and transparent analysis. The system's architecture ensures strong isolation between the malware execution and the analysis machine, while providing complete visibility into system memory. Our implementation demonstrates significant performance improvements and confirms the feasibility of integrating hardware-assisted memory forensics into bare-metal malware analysis workflows. Future work includes extending support to additional executive object types and adopting network-based communication to improve scalability across multi-system environments.

CRediT authorship contribution statement

Dan Cristian Turicu: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Formal analysis, Conceptualization; **Florin Oniga:** Writing – review & editing, Validation, Supervision, Methodology, Formal analysis, Conceptualization.

Data availability

No data was used for the research described in the article.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by "Romanian Hub for Artificial Intelligence-HRIA" project, Smart Growth, Digitization and Financial Instruments Program, MySMIS no. 351416.

References

[1] Bulazel A, Yener B. A survey on automated dynamic malware analysis evasion and counter-evasion: pc, mobile, and web. Proceedings of the 1st reversing and offensive-oriented trends symposium. 2017.
[2] Chen X, Andersen J, Mao ZM, Bailey M, Nazario J. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. IEEE international conference on dependable systems and networks with FTCS and DCC. 2008;10–177. https://doi.org/10.1109/DSN.2008.4630086
[3] Paleari R, Martignoni L, Roglia GF, Bruschi D. A fistful of red-pills: how to automatically generate procedures to detect CPU emulators. Proceedings of the USENIX workshop on offensive technologies. 2009.
[4] Lindorfer M, Kolbitsch C, Comparetti PM. Detecting environment-sensitive malware. Proceedings of the 14th international conference on recent advances in intrusion detection. 2011;20–338.
[5] Willems C, Hund R, Fobian A, Felsch D, Holz T, Vasudevan A. Down to the bare metal: using processor features for binary analysis. Proceedings of the 28th annual computer security applications conference. 2012;10–189. https://doi.org/10.1145/2420950.2420980
[6] Balzarotti D, Cova M, Karlberger C, Kruegel C, Kirda E, Vigna G. Efficient detection of split personalities in malware. Proceedings of the network and distributed system security symposium. 2010.
[7] Brengel M, Backes M, Rossow C. Detecting hardware-assisted virtualization. Proceedings of the international conference on detection of intrusions and malware, and vulnerability assessment. 2016;21–207. https://doi.org/10.1007/978-3-319-40667-1_11
[8] Garfinkel T, Adams K, Warfield A, Franklin J. Compatibility is not transparency: vmm detection myths and realities. Proceedings of the 11th USENIX workshop on hot topics in operating systems. 2007;6–1.
[9] Rekall Team. WinPmem - a physical memory acquisition tool. 2020 Version 4.0 RC2 [software]; https://github.com/Velocidex/WinPmem.
[10] Moonsols. DumpIt. 2011 Version 1.3.2 [software]; https://www.moonsols.com.
[11] Carrier BD, Grand J. A hardware-based memory acquisition procedure for digital investigations. 2004;11–50. https://doi.org/10.1016/j.diin.2003.12.001
[12] Petroni NL, Fraser T, Molina J, Arbaugh WA. Copilot - a coprocessor-based kernel runtime integrity monitor. Proceedings of the USENIX security symposium. 2004.
[13] Wang J, Zhang F, Sun K, Stavrou A. Firmware-assisted memory acquisition and analysis tools for digital forensics. In: Proceedings of the IEEE international workshop on systematic approaches to digital forensic engineering. 2011;5–1. https://doi.org/10.1109/SADFE.2011.7

[14] Frisk UK. PCILeech - Direct Memory Access (DMA) Attack Toolkit. 2025 Version 4.19 [software]; https://github.com/ufrisk/pcileech.
[15] Volatility Foundation. Volatility 3 Framework: Advanced Memory Forensics Framework. 2025 Version 2.26.0 [software]; https://www.volatilityfoundation.org.
[16] Rekall Team. Rekall Memory Forensic Framework. 2017 Version 1.7.1 [software]; https://github.com/google/rekall.
[17] Turicu DC, Cret O, Vacariu L. Storage Mirroring for Bare-Metal Malware Analysis on FPGA Devices. International Conference on Field-Programmable Technology. 2019. https://doi.org/10.1109/ICFPT47387.2019.00061
[18] Bauman E, Ayoade G, Lin Z. A survey on hypervisor-based monitoring: approaches, applications, and evolutions. 2015;1–33. https://doi.org/10.1145/2775111
[19] Amd. UltraScale Devices Integrated Block for PCI Express Product Guide. 2025. https://docs.amd.com/r/en-US/pg213-pcie4-ultrascale-plus.
[20] Amd. Vivado Design Suite. 2023 Version 2023.2 [software]; https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html.
[21] Amd. Alveo U50 Data Center Accelerator Card Data Sheet (DS965). 2023. https://docs.amd.com/r/en-US/ds965-u50.
[22] Russinovich M. RAMMap - Sysinternals. 2025 Version 1.61 [software]; https://learn.microsoft.com/en-us/sysinternals/downloads/rammap.
[23] Russinovich M. Process Explorer - Sysinternals. 2025 Version 17.06 [software]; https://learn.microsoft.com/sysinternals/downloads/process-explorer.
[24] Hollander R. RAMspeed 1.0.0 Beta for Windows. 2008 Version 1.0.0 Beta [software]; https://www.softpedia.com/get/System/Benchmarks/RAMspeed.shtml.
[25] N.C.S. Trade. GFX Memory Speed Benchmark. 2024 Version 1.1.22.24 [software]; https://www.techspot.com/downloads/6767-gfx-memory-speed-benchmark.html.
[26] Hongyi L, Jiarong X, Yibo H, Danyang Z, Srinivas D, Ang C. Remote direct memory introspection. In: Proceedings of the USENIX conference on security symposium. 2023;18–6043.
[27] Spensky C, Hu H, Leach K. Lo-phi: low-observable physical host instrumentation for malware analysis. Proceedings of the network and distributed system security symposium. 2016. https://doi.org/10.14722/ndss.2016.23121
[28] Kirat D, Vigna G, Kruegel C. Barebox: efficient malware analysis on bare-metal. In: Proceedings of the 27th annual computer security applications conference. 2011;10–403. https://doi.org/10.1145/2076732.2076790
[29] Kirat D, Vigna G, Kruegel C. Barecloud: baremetal analysis-based evasive malware detection. In: Proceedings of the 23rd USENIX conference on security symposium. 2014;15–287.