Runtime Memory Introspection for Behavioral System Monitoring Using DPUs

Liwaa Albanna, Samia Choueiri, Elie Kfoury, Jorge Crichigno
University of South Carolina, Columbia, South Carolina

University of South Carolina
Molinaroli College of Engineering and Computing

Abstract

- Modern malware analysis techniques rely on a range of approaches, including static analysis, dynamic execution, and memory inspection. While these methods provide valuable insights, they often struggle to capture transient runtime behavior or introduce significant overhead.
- Malware interacts with host memory by modifying processes, threads, and modules, leaving behavioral traces that are difficult to observe using traditional techniques.
- This work presents a runtime memory introspection approach using NVIDIA BlueField DPUs and the DOCA App Shield framework. The system continuously extracts structured behavioral features from host memory, enabling low-latency and non-intrusive monitoring of system activity.
- This approach provides a scalable foundation for real-time behavioral analysis and future anomaly detection.

Related Work

- BlueGuard proposes a DPU-based framework for host and guest memory introspection, enabling low-latency and scalable monitoring through hardware acceleration. Its focus is on improving introspection performance and infrastructure rather than building a runtime monitoring pipeline [1].
- RDMI rearchitects memory introspection using RDMA NICs and programmable hardware to achieve high-frequency monitoring of kernel memory with minimal CPU overhead, emphasizing low-level hardware mechanisms without addressing system-level monitoring workflows [2].
- vDefender presents a hypervisor-based system for malware detection using behavioral logs and machine learning, relying on intrusive breakpoint-based tracing and operating as an on-demand analysis framework [3].
- This work focuses on continuous runtime monitoring using DOCA App Shield on BlueField DPUs, emphasizing lightweight feature extraction and performance-aware execution for real-time behavioral observation.

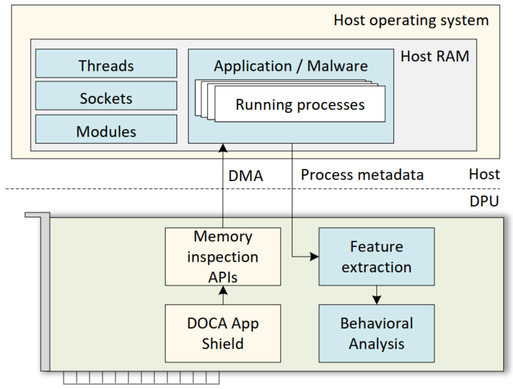
[1] Orenbach et. al., "BlueGuard: Accelerated Host and Guest Introspection Using DPUs," 2025.
[2] Liu et. al., "Remote Direct Memory Introspection," 2023.
[3] Gaur et. al., "vDefender: An explainable and introspection-based approach for identifying emerging malware behaviour at hypervisor-layer in virtualization environment," 2024.

Acknowledgement

- This work was supported by the National Science Foundation (NSF), Award 2417823, 2403360.

System Architecture

- The proposed system leverages a BlueField DPU to perform external memory introspection on a host machine. The host operating system executes user applications, including potential malicious processes, whose runtime state resides in system memory.
- The DPU accesses host memory through direct memory access (DMA), enabling non-intrusive observation without interfering with host execution. Using DOCA App Shield (APSH) APIs, the system retrieves structured information from memory, including processes, modules, and threads.
- The extracted data is then processed through a lightweight feature extraction stage, where relevant behavioral attributes are collected. These features form the basis for monitoring and analysis of system behavior over time.
- The overall design enables continuous, low-latency observation of host memory activity while maintaining isolation between the monitoring system and the host.



Overview of the runtime memory introspection architecture using BlueField DPU and DOCA APSH

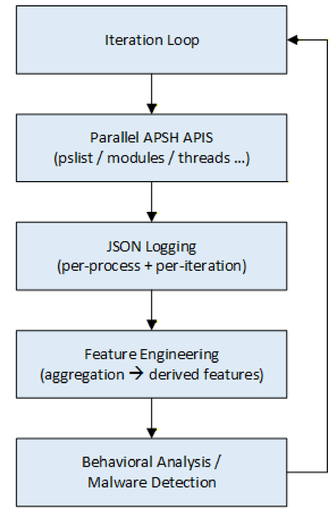
Challenges and Future Work

- Limited compatibility with newer Windows versions, leading to unsupported APSH samples.
- Some features are only available on Linux, restricting cross-platform functionality.
- High-latency APIs (e.g., handles, libraries) require per-process traversal, making them unsuitable for continuous monitoring.
- Certain system features are not exposed through DOCA APSH APIs, limiting observable behavior.

- Apply the monitoring pipeline to real malware scenarios to study memory-based behavioral changes.
- Integrate extracted features with detection mechanisms for anomaly identification.
- Improve performance through refined API scheduling and selection strategies.
- Extend support across different operating systems and expand feature coverage.

Runtime Monitoring Pipeline

- The system operates as a continuous monitoring pipeline that extracts behavioral features from host memory using DOCA App Shield APIs.
- Each iteration captures the current system state through structured data collected from multiple system components.



The pipeline enables continuous, lightweight, and non-intrusive monitoring of system behavior in real time

Execution Model

- The system uses a performance-aware execution model based on the varying cost of memory introspection APIs. Lightweight APIs (e.g., pslist, modules) support frequent monitoring, while higher-cost APIs (e.g., threads, libraries, handles) introduce significant latency.
- To address this, the system employs asynchronous execution, allowing tasks to run independently without blocking. This ensures continuous updates of fast features while slower operations execute in parallel.
- The model prioritizes responsiveness by maintaining frequent access to lightweight features and selectively invoking more expensive operations.

APSH API execution cost per iteration

Memory-Derived Behavioral Features

- The table summarizes features extracted from host memory using DOCA APSH APIs at both process and system levels.
- Raw fields are transformed into aggregated behavioral features (e.g., counts, maxima, timing) that capture runtime activity patterns.
- These features enable detection of anomalies such as unusual threading, module injection, privilege abuse, and resource misuse during live monitoring.

Memory-derived behavioral features extracted from DOCA APSH APIs for runtime monitoring

API	Raw Fields (Per Process)	Derived Features (Per Iteration)	Behavioral Meaning
pslist	pid, name, cpu_time	process_count, total_cpu_time, max_cpu_time, elapsed_ms	Injected / unexpected modules
modules	module_name	module_count, elapsed_ms	Activity spikes / CPU anomalies
threads	module_name	total_threads, max_threads_per_process, process_count, elapsed_ms	Abnormal threading behavior
libs	pid, name, lib_path	total_libs, max_libs_per_process, process_count, elapsed_ms	DLL injection / unusual dependencies
privileges	pid, name, privilege_name, enabled_flag	total_libs, max_libs_per_process, process_count, elapsed_ms	Privilege escalation / debug abuse
handles	pid, handle_count	total_handles, max_handles, elapsed_ms	Resource abuse / handle anomalies