DFIR Memory Analysis – Comparing Windows & Linux

Taz Wake | Halkyn Consulting Ltd | Originally published 2023-12-01

Introduction

The management of user-mode process address space significantly influences operating system architecture, affecting aspects ranging from application execution to cybersecurity. This article offers an examination of these mechanisms, comparing the Linux Kernel and Windows 11. It focuses on the Process Environment Block (PEB) in Windows and its equivalent in Linux, alongside exploring the distinction between user and kernel modes in both systems.

The Process Environment Block (PEB) in Windows 11

In Windows 11, the PEB is a user-mode structure integral to process execution. Key components include:

- **Heap Management**: The PEB holds pointers to the process heap for dynamic memory allocation.

- **Execution Context**: Stack limits, execution start points, and program counters are encapsulated here.

- **Dynamic Link Libraries (DLLs)**: Crucial for managing DLLs, the PEB tracks DLLs loaded into the process. The three main lists - **InLoadOrderModuleList**, **InMemoryOrderModuleList**, and **InInitializationOrderModuleList** - provide an ordered view of these libraries.

- **API Interactions**: Functions like **LdrEnumerateLoadedModules** and **EnumProcessModules** enable programmatic interaction with these lists.

Linux's Distributed Approach

Linux disperses process-related data across various kernel and user-space structures:

- **task_struct**: Residing in the kernel space, it represents the process at the kernel level.

- **mm_struct**: This kernel structure manages memory regions, akin to some PEB functionalities.

- **User Space Data**: Here, environment variables and command-line arguments are stored, accessible by the process.

User-Mode vs Kernel-Mode

The distinction between user-mode and kernel-mode operations in Linux and Windows is a fundamental aspect of their respective architectures. Each system has its own way of implementing the separation between zones, largely reflecting the different ways the two environments were designed.

Windows

- **Design Philosophy**: The separation in Windows is primarily security-centric. It restricts user-mode applications from accessing core system operations directly. This is crucial for maintaining system stability and security.

- **Practical Implications**: For instance, when a user-mode application needs to perform a task like accessing hardware or managing file systems, it must use system calls. These calls are meticulously controlled and mediated by the Windows NT kernel, ensuring that user applications do not overstep their bounds.

- **APIs and System Calls**: Functions such as **ReadFile** or **WriteFile** in the Windows API are examples of user-mode APIs that internally call their kernel-mode counterparts, providing a secure and controlled way of accessing system resources.

Linux

- **More Pronounced Separation**: Linux enforces a more strict and clear-cut separation between user and kernel modes. This is evident in the way memory and processes are handled in the system.

- **System Calls as Bridges**: In Linux, system calls (like **read()**, **write()**, **open()**, etc.) are the primary mechanism through which user-mode applications interact with the kernel. For instance, when a user-mode process in Linux needs to read a file, it invokes the **read()** system call, which then transitions into kernel mode to access the file system.

- **Context Switching and Security**: This transition involves a context switch, which is a key security feature in Linux. The context switch ensures that the user-mode process operates in a limited privilege environment, preventing it from directly accessing or modifying kernel memory or performing unauthorised hardware interactions.

In both operating systems, this separation serves to prevent user applications from performing operations that could destabilise or compromise the system. However, the nature and implementation of this segregation differ, reflecting the unique security models and architectural philosophies of Windows and Linux.

Practical Example: DLLs vs Shared Libraries

The management of Dynamic Link Libraries (DLLs) in Windows and shared libraries in Linux highlights the contrasting approaches of these operating systems:

Windows and the PEB's DLL Lists

- **InLoadOrderModuleList**: This list plays a critical role in the sequential loading of DLLs. It maintains the order in which DLLs are loaded into the process's memory, which is important for resolving dependencies as later-loaded DLLs may depend on earlier ones.

- **InMemoryOrderModuleList**: Sorted by the memory address at which each DLL is loaded, this list is pivotal for memory management and efficient address space utilisation. It helps the system and debugging tools quickly locate and reference DLLs in the process's memory.

- **InInitializationOrderModuleList**: This list indicates the order in which the initialisation routines of the DLLs are executed. This sequence is critical for ensuring the proper setup of the DLLs, as some may require initialisation in a specific order to function correctly.

Linux and Shared Library Management

- **Contrasting Approach**: Linux does not have a direct analogue to the PEB's structured DLL lists. Instead, it employs a combination of filesystem-based and runtime dynamic management for shared libraries.

- **/proc/[pid]/maps**: This file provides a snapshot of the process's memory map, including loaded shared libraries. While it does not explicitly sort these libraries in any particular order, it allows users to view where each library is loaded in memory, serving a similar purpose to the **InMemoryOrderModuleList**.

- **Dynamic Loading APIs**: Linux utilises **dlopen** and **dlclose** for runtime dynamic loading and unloading of shared libraries. These functions do not inherently maintain a specific load or initialisation order like Windows' lists. Instead, they offer on-demand loading, with the calling application responsible for managing the order and dependencies.

- **ld.so**: Linux's dynamic linker/loader **ld.so** handles the linking aspect of shared libraries during process startup. It resolves dependencies and links the necessary libraries from the filesystem, akin to the collective functionalities of the Windows DLL lists.

Cybersecurity and Incident Response Considerations

In the Windows ecosystem, the PEB is a critical structure for cybersecurity analysts to monitor. Given its central role in managing process-specific information, including DLLs, the PEB can be a prime target for exploitation. For example, DLL injection, a common code injection technique, can manipulate the PEB's DLL lists. Analysts often look for anomalies in these lists, such as unexpected or unknown DLLs, which could indicate that malicious code has been injected into a process. Additionally, discrepancies in the order of the **InLoadOrderModuleList** or unauthorised modifications in the **InMemoryOrderModuleList** and **InInitializationOrderModuleList** can be tell-tale signs of tampering.

In Linux, the **task_struct** and memory mappings provide essential insights for cybersecurity professionals. Unlike the structured approach of the PEB in Windows, Linux's more distributed architecture requires analysts to focus on different aspects. Alterations in the **task_struct**, such as unexpected changes in process privileges or modifications in the memory management fields, can signal potential intrusions or exploits. Furthermore, the **/proc/[pid]/maps** file, which details the memory mappings of processes, is a key area for investigation. Irregularities or unauthorised mappings, especially those linking to suspicious shared libraries or binaries, can be indicative of a security breach. Analysts might also monitor the dynamic loading of shared libraries using **dlopen** and **dlclose** to detect runtime injection or manipulation of these libraries.

Practical Implications for Incident Response

In both operating systems, understanding the nuances of how processes manage their address space and related structures is crucial for effective incident response. This knowledge allows cybersecurity analysts to detect, analyse, and respond to various forms of code injection and other memory-based exploits. By closely monitoring these structures and understanding their normal behaviour, analysts can identify and mitigate threats more effectively, ensuring the security and integrity of both Windows and Linux environments.

Summary

The divergent approaches in handling user-mode process address space between Linux and Windows reflect their distinct architectural philosophies. Windows centralises process-related data in the PEB, whereas Linux opts for a dispersed model. These differences necessitate nuanced strategies in cybersecurity and incident response, underlining the importance of a thorough understanding of these operating systems' internals for professionals in the field.

About Halkyn Consulting

Halkyn Consulting Ltd is a UK-based security and risk management consultancy providing 24/7 incident response, digital forensics, threat hunting, virtual security leadership (vCISO) and security compliance support to organisations worldwide. We deliver clear, practical, vendor-independent advice based on real-world threats — no hype, no upsell.

More free security resources are available at www.halkynconsulting.co.uk/security-resources.

Web: www.halkynconsulting.co.uk | **Email:** info@halkynconsulting.co.uk | **Tel:** +44 1244 940858